Vrije Universiteit Brussel – Belgium
Faculty of Sciences
2001

Shagger: a Distributed File-sharing Community with Transparent User Management and Adaptive Replication.

A thesis submitted in fulfilment of the requirements for the degree in Computer Science.

By: Wim Boffé

Promotor: Prof. Theo D'Hondt (Vrije Universiteit Brussel)
Advisors: Johan Fabry and Werner Van Belle (Vrije Universiteit Brussel)

Acknowledgements

I would like to use this occasion to thank some people who supported and helped me while I was working on this thesis.

First of all I would like to thank Johan Fabry and Werner Van Belle who lend me their minds and their time. They put in a lot of work advising me on the writing of this thesis and on solving problems we encountered. Proofreading my thesis was probably not a pleasant thing to do so Johan and Werner, thanks for the efforts you both put in.

Serge Di Marcantonio designed the Shagger logo on the main page.
Great job Serge!

Davy Pierson is a good friend I met at the VUB. Together we talked a lot about possible improvements in file-sharing communities. Looks like we didn't waste all those hours after all. Thanks for the inspiration Davy.

I would also like to thank Tom Mens and Saartje Brockmans for proofreading and commenting on my thesis.

Finally I would like to thank the people that gave me the opportunity to pursue my interests, my parents.

Content

Content 1
Part I – File-sharing Communities 5
1. Introduction 5
1.1 Typical Use 6
1.2 Current Problems and Restrictions 6
1.3 Goals 8
2. Current File-sharing Communities 10
3. Napster 12
3.1 Network Structure 12
3.2 Data Storage 12
3.3 User Management 13
3.4 Searching 13
3.5 File Replication 13
3.5 Problems and Disadvantages 14
4. Kazaa 15
4.1 Network Architecture 15
4.2 Searching 15
4.3 File Replication 15
4.4 Problems and Disadvantages 16
5. Gnutella 17
5.1 Network Structure 17
5.2 Data Storage 18
5.3 User Management 18
5.4 Searching 18
5.5 File Replication 20
5.6 Problems and Disadvantages 20
6. Overview 21
7. Conclusion 22
Part II - The Shagger Way 23
1. Introduction 23
2. Shagger Network Architecture 25
2.1 Introduction 25
2.2 Properties of the Shagger Network Architecture 25
2.3 Levels 26
2.4 Building the Network 28
2.4.1 Action Radius 32
2.4.2 Connection of a New Node 33
2.4.3 Reconnecting a Node 36
2.4.4 Post-(re)connection Tasks 39
2.4.5 Disconnection of a Node 40
2.5 Conclusion 44
3. Data Storage 45
3.1 File Chunks 45
3.2 Shared Files List 46
3.3 Index List 47
3.4 User Data 48
3.5 Statistical Data 48
3.6 Data Location 49
3.7 Conclusion 49
4. User Management 50
4.1 Influencing the Behaviour of a User 51
4.2 Ability per Level 52
4.3 Conclusion 52
5. Searching 53
5.1 GUID-search 53

5.2 Identity-search 55
5.3 Fuzzy-search 56
5.4 Conclusion 57
6. File Replication 58
6.1 Statistical Data 60
6.2 Flushing 65
6.3 User Data 65
6.4 Conclusion 66
7. Downloading 67
7.1 How to Find Results 67
7.2 Determine Download Locations 67
7.3 Downloading 68
8. Error Recovery 69
8.1 Dirty Node Disconnection 69
8.1.1 Dirty Disconnection of a Level Zero Node 69
8.1.2 Dirty Disconnection of a Node of Level One or Higher 69
8.2 Loss of User Data 71
9. Future Work 72
9.1 External Referencing 72
9.2 Self Served Pages 72
9.3 File Requests 73
10. Problems and Restrictions 74
10.1 Initial Connection to the Network 74
10.2 Overloading the Network 74
10.3 Community Overview 74
10.4 New Files Online 75
11. Overview, Evaluation and Conclusion 76
11.1 Overview 76
11.2 Evaluation 78
11.3 Conclusion 80
Bibliography 81

Part I – File-sharing Communities

1. Introduction

Usage of the Internet is currently evolving toward community-based private networks. There are already many applications available that gather people who share the same interests into so-called communities. This is achieved by offering the user extra facilities that align with his or her interests. Examples of these facilities are: chat, FTP, instant messaging, and so on.

Internet communities grew out of the idea that every PC connected to the Internet is a possible provider of resources. However, setting up a web or ftp-server is not a trivial thing to do for most users. Therefore the procedure of publishing resources had to be made simpler. Community applications were developed to lower the threshold of publishing and obtaining resources. This is accomplished by offering the users extra features that make publishing resources very straightforward. Although this is a big advantage for the users, the community also introduces a lot of new problems: whereas the 'normal' content providers of the World Wide Web are dedicated machines, now machines are being used to serve resources that were not meant to be online all the time, such as the users home PC. This causes the resources to be more dynamic than e.g. WWW content. Search engines that search the resources on a community network will have to apply a different strategy than search engines used for finding WWW-related content.

File-sharing communities only operate well if many users are connected to the community network. This is because as more people connect to the network, more resources will become available. Therefore file-sharing communities are mostly used on Wide Area Networks (called WAN from now on). WAN, like the Internet, offer users the possibility to communicate with a vast amount of other users connected to that network. Although this might seem a good thing, most of the problems with Internet communities are caused by WAN. Some characteristic problems of WAN include host unreliability, non-constant and low bandwidth, and network congestions. Current Internet community applications deal with these problems in a very straightforward fashion, not utilizing the community advantage to its full extent. For example: A basic service every file-sharing community offers is transfer of files. Problems with file transfers can be caused by WAN. Community clients implement file transfers as FTP-sessions. They don't use the amount of bandwidth offered by other nodes on the community network. By downloading from all other users that share the file the transfer speed would be increased.

One kind of Internet community is the file-sharing community. People who are interested in software, music, or other resources available as files use this kind of community to find or publish these files. In most cases, the interaction of the users with the rest of the community is done through a specialized community application. These applications take care of all community related facilities like user authentication, file search and up-and download management.

In this thesis we will concentrate on finding new ways to manage file-sharing communities and also apply network resource management strategies to file-sharing communities. Aside from network-related research, extra services will be proposed to improve the overall effectiveness of the file-sharing community.

1.1 Typical Use

In this chapter we explain what file-sharing applications are typically used for. It is clear that file-sharing applications are used to distribute data. We will therefore first discuss what kind of data is spread using file-sharing communities. Then we will spend attention on how users use a file sharing community and to what purpose.

Current file-sharing applications are not used to search for information on E.g. how to bake an apple pie. Such information is typically contained in a text file or even a HTML-file. To find such information in a file-sharing community, the filename would have to describe the exact content of the file itself.

On the other hand, through the web, information like this is easy to find: use a search engine to search for an online cookbook and then look for the words "apple pie". Using the search results you will be able to obtain a document that contains information on how to bake an apple pie.

Most of the file-sharing applications we will discuss in this text offer the user an enormous amount of data. When looking at the kind of data there is demand for mostly media files. This is because of the ease with which these files can be identified. For example a music file from the artist called "Queen" with title "Bohemian Rhapsody" is uniquely identified by these values. Variations are still possible e.g. a live version of the number or a Karaoke version, but in the end the identification leads to results that are almost perfect matches. This kind of easy identification is what makes a file-sharing community so easy to use.

Files shared on these communities are therefore bound to the rule of being identifiable by their filename. These files are mostly media files as their abide the previous rule. Media files can be split up into different types e.g. video files, music files, etc.

File-sharing communities contain a vast amount of data. Users use these file sharing communities to obtain that data. The typical procedure to do so is:

- Open the community application
- Define a search query to identify the data searched for.
- Submit the search query.
- Select the files to download from the results returned.
- Download the files.
- Close the community application.

A user typically uses a file-sharing application to get in touch with other people that are interested in media files.

1.2 Current Problems and Restrictions

File-sharing communities are dealing with a lot of problems that are specific to Internet communities. We will discuss these problems and those caused by using the Internet as a way of communication.

One of the major drawbacks of the established file sharing communities is, that when a user decides to go offline or loses his network connection, his shared files become unreachable to the entire network. We will propose a solution for this, and evaluate its usability.

It is clear that for the network to be able to offer files, users must be willing to share their resources. Therefore the assumption: Users share files, will have to be made. This makes the available resources dependent on the user's willingness to share with the community.

Current file-sharing communities do not give the user an idea about what is socially acceptable. This is because of the context of the social interaction in file-sharing communities. The interactions are very brief with no long-term view. If we were to provide ways to enable different users to communicate on a regular basis, these users would have a better idea of the social context they are operating in. The social context of file-sharing communities has been neglected until now. However, we are convinced of the added value of having more interaction between users in the community. When users can communicate more easily, more social context is given to the community. If a user has a better understanding of who is downloading his files and what this user is offering in return, less selfish behaviour will be displayed. When users have the impression that everyone is taking advantage of their resources, they will share less files. This way, users become "selfish" and refuse to contribute to the network.

This theory can be illustrated through the behaviour people expose on Internet Relay Chat (IRC). IRC is primarily used as a communication tool. However it is also used to get in touch with people that share the same interests. People that are interested in finding new media files use IRC to trade these files. Users get a direct pay-off for sharing their files (by trading) on IRC. They will therefore be motivated to offer their own resources in return.

There are ways of motivating the user to share files, and we will discuss some of these techniques and their effectiveness. In the end a user must still be able to determine which files he wishes to share.

Present network architectures used for community applications, such as client-server architectures, do not scale by themselves [1], which limits their size and rate of expansion. When a server needs to serve ten clients, the machine needed as server can be an ordinary home computer. If we look at a server serving 10.000 clients, we see a completely different machine with a much higher processing power. When looking at a website with an enormous amount of hits every day (E.g. www.yahoo.com) we see a server farm that uses load balancing to respond to requests made by clients [2]. Servers like these used by yahoo are rarely seen as home machines for a single user. Since we don't want to be dependent on any external organisation but still keep our network scalable, we will therefore have to rule out the case of using a client-server layout as our network architecture. These client-server architectures are also very vulnerable to external attacks because they centralise all critical data.

With today's increasing personal bandwidth, the local connections are no longer the bottleneck of the Internet, but the problem rather lies with the backbones of the Internet. [3] This situation leads to low-speed file transfers. We propose an effective way of dealing with this problem using the properties of Internet communities. The speed of Internet connections at home is increasing which gives us the opportunity to use this bandwidth for the advantage of the community. For example: by downloading a file from multiple clients in parallel. This way the maximum amount of bandwidth is used while downloading a file.

When data is presented to a community network, this data is in no way checked on validity or usefulness. This introduces a vast amount of 'garbage' on the network, wasting community resources. If we would check the validity of a file, E.g. A compressed file can be checked easily through its checksum, we would

already drastically reduce the amount of garbage on the network. Throughout this text we will spend no attention on this optimisation as it can be easily implemented in any file sharing application through implementing these checks in every community client.

As files are shared worldwide, the network architecture as well as the search queries have to take in account high latency times and disruptions in the availability of resources. All community resources are located on machines controlled by users and are therefore dependant on their uptime. These machines are however not designed to have an uptime similar to a web server. This fact, together with the unreliable Internet connections at home and the inexperienced users using the file-sharing community, contributes to this instability.

Network partitioning is one of the biggest problems with distributed file sharing networks. When a network is partitioned, a user that connects to the network, cannot reach the entire network. For example: when a user issues a search query and is not connected to all clients on that network, he will not receive a full list of files that are available on the network. A direct consequence of this is, that not all resources are available for all the users on the network. This is a major problem in the Gnutella network [4]. The Gnutella network and all of its main features are discussed later on in this text.

1.3 Goals

There is still no file-sharing community that offers users everything they need. This statement is proven by the diversity in file-sharing applications available today. Although there are differences between all of the communities, they offer the same basic services to their users. We will outline the goals we think are important to build a "good" file-sharing community.

One of the primary goals of users of a file sharing community is the acquisition of files. It must therefore be possible for a user to download files from the network.

Users must be able to submit files or other resources to the network. Because of this, resources will no longer be dependent on the uptime of the contributing user, but rather on the community itself. This approach introduces the need for extra community resources and a more elaborate management structure for these resources. Whereas the conventional file-sharing communities only keep track of files and their availability, there will now be an interest for other resources besides files offered by a user. For example: disk space, processing power, memory, etc.

Users will have to be made conscious of the value of community resources. This is the only way to improve the quality of the entire community. Therefore users will receive an advantage for submitting 'valuable' resources to the network and obtain a disadvantage by wasting resources. To be able to tell if a resource is valuable, users must be able to rate resources on their usability and quality. This way, the quality of the entire network will increase because of the fewer files shared that take up valuable resources and have no community value. This kind of adding metadata to files can also be exploited in commercial terms to add another layer of functionality to the community.

One of the most important properties of the new network design should be an effective way of scaling. The network should scale without getting partitioned, losing performance and creating overhead on network complexity or compromising the vulnerability of the network.

The community will not be managed by any superior entity. This helps to make the community more robust and persistent as a whole. Note that this rules out client-server architecture that most of the current communities have chosen as their network architecture.

There has to be a fast way of searching for resources and there will have to be a well-defined list of matches. Searches must be possible throughout the complete network and a user must be able to tell whether the community is fully searched. Searches should however not put a to great strain on the community resources.

Since the main purpose of a community network is to provide data that is available to every user on the network, no effort will be made to provide security on the resources that are shared between users. There will however be some kind of identification of users, to prevent misuse of the network.

As a final constraint the network functionality and resource allocation must exist completely invisibly to the community members. All the decisions they will have to make will be proposed by the community system itself. This will ensure proper load balancing of the network and the optimal spreading of information and data over the network.

2. Current File-sharing Communities

In the remainder of part one we will discuss some examples of file-sharing communities. These examples have been chosen because of the different ways problems are dealt with. The properties that are evaluated for every file-sharing community are discussed in this chapter.

We will evaluate the following properties for every community:

- **Network structure**
  Network structure or architecture is important to a file-sharing community as it determines its robustness and dependency on external organisations.
- **Data Storage**
  With data storage we mean the kind of data and the place where data is located on the network.
- **User Management**
  User management is the way a file-sharing community handles its users. For example: are users free to roam or are they bound to certain machines? Are all users treated equal?
- **Searching**
  A very important property of a file sharing community is the way it handles searches. For example: can users determine if the complete network has been searched? What can a user specify in his search query: a filename to look for, a file type and/or a file size or a natural language query?
- **File replication**
  File-sharing communities have to offer replication. Replication improves the availability of files and offers the possibility of performing load balancing. We will look at the different ways current file-sharing communities handle replication.
- **Error Recovery**
  Every file-sharing community has to implement some kind of error recovery. We will look at when error recovery is needed and how errors are dealt with. The most interesting case of error recovery we will discuss is the case of an unexpected node disconnection.

We will evaluate Napster, Kazaa and Gnutella by these properties. Together with their differences we will also discuss their problems and disadvantages.

Before introducing the first file-sharing community, we first formalise some of the concepts used in the discussion of the examples.

Node:
A node is bound to a fixed machine, including that machine's network connection. If the network connection changes, the node is treated as being different.

Resource:
A resource is supplied to the network by a node. Storage, processing power, shared files and bandwidth are the main resources a node can offer to the network.

The community:
The community is the set of all the nodes connected to the network at any given moment in time.

Connection:

A connection is a network connection that is maintained between two nodes.

Uptime:
The term uptime is used to describe the amount of time a node is operational. With operational we mean: the node is connected to the network and it is responsive to requests from other nodes and to requests from its user.

The network:
The set of nodes and the connections they share makes up the network.

We now have a more formal understanding of the terms used in file sharing communities and we can start with evaluating the example file-sharing communities: Napster, Kazaa and Gnutella.

Lately, file-sharing applications have been the subject of some hype. The most widely known file-sharing application is called Napster [5]. Napster was the first well-known application that used resources (processing power, disk space, and memory) from computers connected to the Internet to share files amongst music lovers in a Peer-to-Peer (P2P) [6] fashion. P2P is a technology that has received a lot of attention with the introduction of file-sharing communities.

There were already other applications around that used distributed resources to perform tasks other than file-sharing. These applications primarily use computing power of client computers connected to their network. However, the application wasn't immediately useful to the owner of the machine. For example: SETI@Home [7] is an application that uses shared resources to Search for ExTraterrestrial Intelligence. Users willing to install the client program allow SETI to use computing cycles from their computers. SETI uses this computing power to analyse radio telescope data.

The revolutionary thought in Napster was not the idea of using resources spread across the Internet to its advantage. What was new, was the idea to gather people that loved music around one application that offers:

- Search engine: dedicated to finding mp3 files only
- Internet Relay Chat (IRC): a way to find and chat with other MP3 users while online
- File-sharing: the ability to trade MP3 files between users, without having to use a centralised server for storage

In the rest of this chapter we will evaluate the Napster community by the properties introduced in the previous chapter.

3.1 Network Structure

The Napster network is structured according to the standard client-server network layout. This means there is one central server and clients connect to this server. A client is a pc with the Napster application installed.

3.2 Data Storage

The different kinds of data on the Napster community network are:

- Shared Files
- User Data
- Shared Files Index

The shared files are the files users share to the Napster community.

User data is a user's name, password, email and information on the user's machine. This last piece of user data is e.g. the machine's Internet connection type, such as ADSL, cable, etc.

The shared-files index is maintained on the central Napster server. Search queries are executed on this shared-files Index. Note that the way a client connects to the Napster network implies that the index list is updated when clients connect.

Connection of a client to the community is according to the following sequence:

1. The client connects to the server.
2. If the user is not registered on the server the user has to enter a username and password. This username and password are stored on the server.
3. The user logs in to the server.
4. The client sends an index of all files located in the shared directory of the client to the server. This index is updated on a regular basis.
5. The user is connected to the network.

The overall result of this mechanism is that the server has a complete list of all files shared by all clients connected to the network. The server can easily determine what files are available at any given time.

3.3 User Management

In the Napster community every user gets a unique username. This username identifies the user and not the machine that user works on. This means that users can easily roam from one machine to another without losing their username. The Napster network only provides this username for user-friendly identification of users. Information on the behaviour of the user is not used to improve the network's effectiveness. All users on the Napster community are treated equal.

3.4 Searching

Searches on the Napster network are performed on file names. Therefore searches are simple sub-string matches on all the names of the shared files.

Searching the community for a certain file is reduced to the following simple sequence of actions:

1. The client sends a search query to the server.
2. The server executes this query on the list of files that are online.
3. A response from the server is sent to the client containing references to all the files that matched the query.

Downloading a file from the network is done in a peer-to-peer fashion: a direct connection is established from the client that wants to download a file to the client that shares the file. Therefore downloads are performed without the intervention of a server.

3.5 File Replication

Through file replication file-sharing communities improve the availability of files. However, on the Napster network there is no explicit file replication. Files that are downloaded are saved in a designated directory. This directory is automatically shared to the community. Therefore, users that download files and automatically share these files anew, implicitly implement file replication.

3.5 Problems and Disadvantages

One of the disadvantages that mark client-server architectures is its vulnerability to Denial of Service (DoS) attacks. A DoS attack is typically not done by only one machine connected to the network. A lot of machines request data from one machine and because of this (un) planned increase in the amount of requests, the server cannot handle all requests. This leads to denial of requests made by clients; services provided by the server are not available anymore. Napster uses a client-server based network architecture and therefore inherits this disadvantage from its architecture.

When a user gets a response to a search query there is no way he can determine the speed with which a file will be transferred. Because of this he cannot select the file that will download the fastest from the results.

The way Napster handles files leads to a list of matches for a search query. The matches returned in this result are not unique. The same file can reside on different machines. This leads to an unclear view on the results of a search query. Many files are listed twice and it is not clear whether they are duplicates or not.

Probably due to a bug in the Napster file-sharing application, file downloads are frequently stopped before they are fully completed. This bug together with the fact that downloaded files are automatically shared again lead to a large number of incomplete files on the network. Users don't mind if the final 2 seconds are missing from their audio file. However, if this file is downloaded again more data will be lost at the end of the file.

Searches are conducted on filenames. Therefore searching only for a type of file is impossible. This is not a real problem as Napster is limited to mp3-files only.

Kazaa [8] is a file-sharing application that was built with Napster as an example. Although Kazaa has many similarities to Napster, we chose to include Kazaa in this text because of the way Kazaa handles shared files and the way it searches for files. Again we will use the properties introduced in the introduction of this chapter to evaluate this file-sharing community. As user management, data storage and replication conform to the way Napster handles these properties, no attention will be spent on them in this chapter.

4.1 Network Architecture

The Kazaa network architecture resembles Napster in multiple ways. There is a central server that indexes shared files on the network and clients are connected to this server. The architecture of the Kazaa network is the same as Napster's architecture.

4.2 Searching

Searching the Kazaa network is similar to the way Napster handles a search request:

1. The search query is sent to the central server.
2. The server executes the query on all shared files.
3. The server sends a response back to the client the search query originated from.

The difference lies with the way Kazaa uses the information that it retrieves from the shared files on the network. Instead of using just the filename to perform search queries on, Kazaa also uses Metadata stored in these files. This Metadata can add context to a file, e.g. an album on which the song was published, which is absent in Napster. This enables Kazaa to perform more exact searches than Napster. Metadata can be added to any file. An advantage of Metadata is the possibility of describing different files according to the same criteria. For example: a music file, video file and even a software package can have an author. Another advantage Metadata offers is the possibility to distinguish between two seemingly identical files.

4.3 File Replication

The way Kazaa handles searches is not the only improvement Kazaa offers over Napster. When a user performs a search on the Napster network, he gets a list of all files that match the query. This list contains duplicate files that are exact copies, but are located on different hosts. The Napster network treats these files as being different. Kazaa however treats all files shared on the network that are exactly the same as one file. These files therefore only occur once in the list of matches returned by the server.

When downloading one of these files, the client application contacts the server and retrieves a list of all clients offering the file. The client then contacts all these

clients that share the needed file and downloads the file from all these clients at once.

Each client offers a different piece of the file for download. This method is proven [9] to be very effective in terms of maximising the bandwidth-usage of the client that is downloading. The amount of a file a client downloads from a host depends on the amount of bandwidth of that host. From a client with a small amount of bandwidth only a small piece of a file will be downloaded. However, from a client with a high bandwidth a larger piece of the file will be downloaded. This way, clients that only have limited bandwidth don't have to sacrifice as much bandwidth to the community as clients with high bandwidth.

4.4 Problems and Disadvantages

Because of the similarities between Napster and Kazaa, Kazaa also inherits the problems of Napster.

However, Kazaa has succeeded in solving some of the problems or disadvantages Napster introduced:

1. **Transfer speed estimation for a file:** Napster offers no estimation as to what speed a file can be downloaded from a client. Kazaa does offer this service to its users. This is done by testing the transfer speed of a client to the server. If a file can be downloaded from multiple clients at once the estimated transfer speed of a file is the sum of all transfer speeds of clients offering the file.
2. **Offering file downloads from multiple hosts:** By downloading files from different hosts Kazaa maximizes the bandwidth usage of the client that is downloading a file.

While we stated back in chapter 3 that Napster was the first well-known file-sharing application to be released, Gnutella [10] actually was the very first file-sharing community. However, usage of Gnutella was too complicated for most users and therefore Gnutella didn't get as much attention as Napster. From a technological point of view Gnutella differs completely from Napster. This makes Gnutella a very interesting example. However, users of the Gnutella file-sharing network notice little difference compared to the Napster file-sharing network except for the ease of use. We will discuss the main technological properties of the Gnutella network and clarify how the network operates.

5.1 Network Structure

On the Napster and Kazaa network, shared files are indexed on a central server. Gnutella is a decentralised network and therefore doesn't use a central server. All Gnutella hosts are called servants because of the lack of a central server.

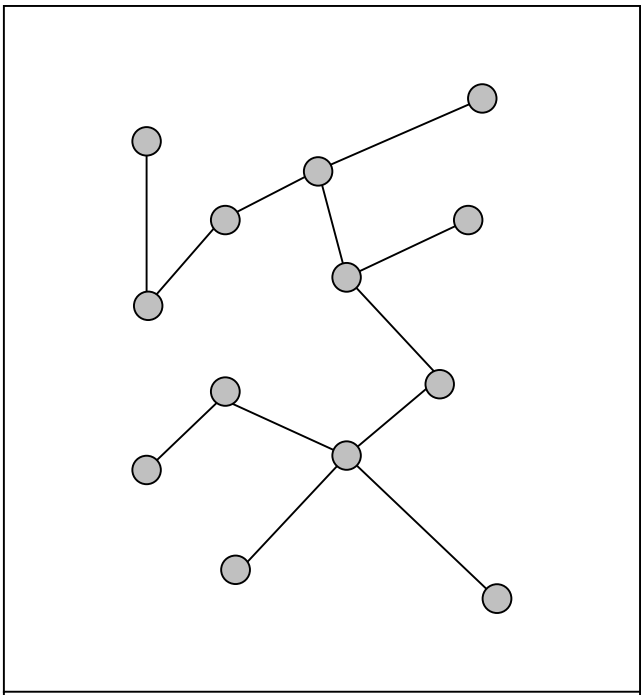


Figure 5.1.1 – An example Gnutella network.

Figure 5.1.1 depicts an example of the Gnutella network. The network is completely flat. Messages between clients are sent over the network from one node to another.

Because of the distributed network architecture of Gnutella, there is no central point where users can connect to the network. Napster and Kazaa both have a central server that also serves a first point of contact for new client applications that want to connect to the network. Therefore users that want to connect to the Gnutella network have to search for an IP-address of a servant connected to the network. There are Internet sites that constantly publish IP-addresses of servants of the Gnutella network.

We will now explain the sequence of actions that are performed for a client to connect to the Gnutella network.

1. The user gets an IP-address from a known web page.
2. The user supplies the Gnutella client application with the IP-address and the client application connects to that address.
3. The servant is connected.

The client is now connected to the network and can receive messages from other nodes. It can also start sending search queries to other nodes.

5.2 Data Storage

As there is no central server that keeps a list of files shared on the Gnutella network, all nodes have to provide their own way of searching their shared files. This approach offers both an advantage and a disadvantage over the central server-approach. By letting every node by itself determine the way it searches his shared files, more matches can be obtained from a search request. For example: when a node receives a search query for an author of a song, it can look up the albums published by that author. This way by searching for an artist all his albums can be found. The disadvantage of this approach is that a user can never know what is going to happen to his search. This could lead to inexact matches that have been deduced from the original search query, but are not related to it in any significant way. For example: searching for the pop group "Queen" could return pornographic material on the queen of England, which was certainly not requested.

5.3 User Management

Gnutella doesn't use usernames to control its users in any way. Therefore there can be no user management. Anyone who wants to connect to the Gnutella network is granted the same rights and can access the same resources. This is a major difference between the way Napster and Kazaa perform user management. The servant is identified by its IP-address and therefore the user cannot easily change machines. However this is not a problem, as the network doesn't store any user information. Only the user's network connection is identified through the machines IP-address. The network uses this IP-address to refer to the servant.

5.4 Searching

Searches on the Gnutella network are more complicated than Kazaa or Napster searches. As there is no central server that can be sent a search query, a search will have to traverse the network in order to find matching results.

1. A servant sends a search query with a given TTL (between 1 and 7) to all the servants that are connected to it.
2. Each of these servants tries to mach this query with the files it shares.

3. If matches are found on one of the servants, the originating node is contacted and sent the matches.
4. Every servant that executed the search query in turn decreases the TTL and sends the query on to every servant he is connected to if the TTL is greater than zero.
5. Every time a servant has found matches to the query amongst his shared files it sends these results back to the servant the query came from. This is continued until the originating servant is sent the results of its search query.

The Time to Live (TTL) ensures that the network is not flooded by searches. A disadvantage of using a TTL is that it is not guaranteed that a node can reach every other node on the network. This also means it is possible that a search query can't search all nodes on the network. Therefore not the entire network is searched and an incomplete list of results is returned. This might seem very bad, but in practice information has the habit of getting around. Figure 5.4.1 depicts what happens in such case.

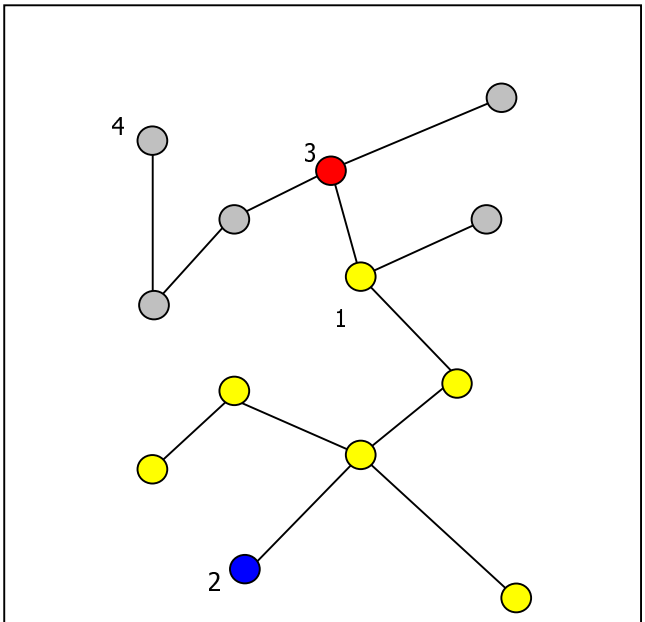


Figure 5.4.1 – Example of a search with TTL = 3

In Figure 5.4.1 the nodes that are reached by a search query from node 2 are colored yellow. Because of the TTL value of three, node 2 can not search the entire network. If the information on node 3 is what node 2 is actually searching for, node 2 will not find the information. In practice however, if the information stored on node 3 is popular, another node (let's say node 1) will download the information. When node 2 issues the same search query after such a download, it will find the information on node 1.

Note that the distribution of nodes on the Gnutella network leads to an unfair situation: node 1 will only have perform search queries with TTL=4 to search the complete network. Node 4 will have to use a TTL=8 to search the entire network.

5.5 File Replication

The Gnutella network handles files the same way as Napster does: every file, even if it is a duplicate is treated as different if it is located on a different servant. File transfers are also conducted in the same manner as on the Napster network: a file is transferred in a peer-to-peer fashion from one servant to another. File replication is therefore implemented in the same implicit way as it is in the Napster file-sharing community.

5.6 Problems and Disadvantages

Gnutella has been proven to be an efficient network, but has problems with scaling [11]. This is demonstrated by the following example: If we state that every node is connected to 6 other nodes, and one of those nodes sends a message with a TTL value of 10, this would result in 6 + 6*5 + (6*5)^2 + (6*5)^3 + … + (6*5)^10 messages to be exchanged. Needless to say, this would put a serious strain on the network. To avoid this the standard TTL value adopted in Gnutella is seven. To protect the network from DoS-attacks, servants that notice high amounts of messages originating from another servant ignore those messages.

However, all the measures taken by developers to protect the network from overloading doesn't change the simple fact that the Gnutella architecture won't scale.

To prove this we use statistics gathered by Clip2 DSS [12]. These statistics show that the network went through 2 major phases.

- A pre bandwidth barrier phase.
- A post bandwidth barrier phase.

The bandwidth barrier is the exact moment in time the Gnutella network started to generate so many messages that modem users could no longer use the network. The bandwidth at the disposal of modem users didn't suffice to accommodate the network traffic generated by messages sent by other servants. Therefore these users became dead ends of the Gnutella network. They were unable to get resources from the network because their internet connections overloaded with messages generated by the network.

6. Overview

This chapter offers an overview of the file-sharing communities we discussed. The different properties that we evaluated for Napster, Kazaa and Gnutella are summarised in the table below to provide a brief overview of their successive strengths and weaknesses.

Community	**Napster**	**Kazaa**	**Gnutella**
Property			
Network architecture	Client-server	Client-server	Distributed
Search speed	Almost constant	Almost constant	Dependant on network extent
Complete list of results	Yes	Yes, but limited to 200 results per search	Dependant on network extent
Replication technique	Implicit	Implicit	Implicit
User management	Centralised	Centralised	Absent
Fairness	Everyone equal	Everyone equal	Unfair opportunities for users (searches)
Data storage	Centralised	Centralised	Decentralised
Scalability	Through external intervention	Through external intervention	Automatically (However, badly implemented)
Implication of node failure	In case of server: network down	In case of server: network down	Network can still function normally
File downloads	From one source	From multiple sources	From one source
Compatibility with file-types	Mp3 only	All types	All types
Level of detail of search results	File size, bit rate, name	Title, author, file size, different sources, etc	File size and name
Download speed estimation	No	Yes	No
Network balancing	Not applicable	Not applicable	No balancing
Adaptation to users	No	No	No
Initial Connection to the network	Through central server	Through central server	Through any connected node

7. Conclusion

In the goals section of chapter 1 we introduced the different properties a "good" file-sharing community should have. We will now discuss which of these properties are met by current file-sharing applications and which need to be added or improved.

None of the example file-sharing communities we discussed are trying to offer their users a better uptime of shared files. Therefore files are still dependant on the users sharing those files. To be able to offer a better availability of files the network will have to provide this service. A possible way to do this would be by setting up a dedicated server that replicates the shared files with the highest demand. This dedicated server must be managed by the organisation controlling the central server to ensure its stability. Although this is a very simple solution, it would already dramatically improve the availability of popular files on the network.

The way Kazaa handles file-transfers is a good example of how to use community properties to improve the service offered by file-sharing communities. File-sharing communities have the property that every client that downloads a file automatically shares that file. The advantage that is deducted from this property is: downloading from every source at once maximizes the amount of bandwidth when downloading a file. Although this is an improvement over the way e.g. Napster handles downloads, there is still a setback: every client from whom the file is downloaded must share the complete file. When music files are shared this does not pose a problem. The size of video files on the other hand is a multiple of that of music files and can therefore not be stored on a user's hard disk for long before this fills up. The solution for this problem would be sharing only a part of the file. This would enable the client to share a file before it has completely finished downloading the file.

User management is close to non-existent in the current file-sharing communities. This prevents the communities from examining the behaviour of their users and therefore file-sharing communities cannot take advantage of the behaviour of their users.

An effective way of scaling should be the main property of a file-sharing community. This is however not always the case as was pointed out in the chapter on the Gnutella network architecture.

The main conclusion that we can draw from part 1 is that current file-sharing applications are not efficiently using community properties. Using these properties to their advantage would improve the quality of the services they can offer their users.

Part II - The Shagger Way

1. Introduction

We will now propose a new architecture for file-sharing communities that addresses the problems discussed in part one. The alternatives we will introduce improve the usability and performance of file sharing communities by intelligently using basic community properties. Major changes include a new network architecture and an alternative to user management in current file sharing networks.

We designed the network architecture with some basic properties in mind. These properties are as follows:

1. No central server: the network has to be completely distributed with no need for a central server.
2. The network should not be dependant on any of the nodes. Therefore failure of a limited number of nodes should not cause the network to stop functioning = no single point of failure.
3. Transfer speeds between nodes should be optimised.
4. The network must scale without external intervention.
5. The community network has to provide a better uptime of shared files than current file-sharing communities offer.

As pointed out earlier none of the file sharing communities we discussed fulfils all of these properties. The Gnutella-network is the closest match, with no need for a central server and adaptation to node failure. However, Gnutella suffers from new problems e.g. with scaling and searching.

Here we'll discuss why we want these properties in our network, and what purpose they serve:

The lack of a central server (1) will make the network more robust, eliminating the single point of failure of the server. Another effect of this property will be that the network can be scalable without intervention (4). A client-server architecture doesn't scale because of the amount of request a server can handle is fixed. The amount of requests the server has to handle on a growing network keeps on rising and therefore external intervention is required.

Making the network completely distributed can make it more robust (2) and less dependant on particular nodes. If we don't store all information in one node the network will have less chance of being disrupted by the failure of a few nodes.
By using data that can be regenerated, the amount of critical data can be reduced. Data can be regenerated if it can be deduced from data available anywhere on the network. If the network cannot regenerate the data, this data is called critical data. For example: when a node fails, another node has to take over his responsibilities. If all information needed by the replacement node is available on the network, the data can be requested from the network. Therefore, disconnection of a node, which contains only data that can be regenerated, can never cause loss of critical information. In chapter 3 "Data Storage" we will introduce the different kinds of data and their location on the network.

By using a new technique in dealing with files shared on the network, a better availability of shared files (5) is achieved and new options arise for transferring files (3). This technique, which depends heavily on some basic properties of file sharing communities, will be discussed in chapter 6 "File Replication".

As a result of the constraints we put on our community network, new problems are introduced in our file sharing community. The biggest problem with distributed network architectures is network management. There is no organisation that is controlling the network; therefore the network will have to manage itself. In chapter 4 "User Management" we will introduce user management that supports and maintains the network architecture to accomplish "self-management".

The chapters 2 "Shagger Network Architecture" and 4 "User Management" will explain the network architecture that we designed followed by the user management policy that is used to maintain this network architecture. The recurring theme throughout part two will be the following behaviouristic axiom:

Users' actions that are beneficial for the network will also yield a direct profit for the user.

2.1 Introduction

The notion of a resource was already introduced in part one. However, the Shagger community makes a distinction between two kinds of resources: user resources, controlled by the users, and network resources, controlled by the network.

This separation between user-and network-resources is important to make, as it very clearly distinguishes Shagger from other file sharing communities. In other file-sharing communities, the users control all the resources. For example: the user determines when to take his files offline, and when to download a piece of data.

With Shagger the user is still in control, but the network can also make these decisions. For the network to be able to do this, it needs resources such as bandwidth and disk space. These resources are allocated by the user of a machine and "donated" to the community. The community can then use these network resources to improve the overall effectiveness of the community.

2.2 Properties of the Shagger Network Architecture.

This part will explain the architecture in terms of the basic properties we had in mind when starting the design.

What we want is a network in which any connected node can reach any other connected node on the network. So, basically the network should offer a logical full-interconnect in normal circumstances. With normal circumstances we mean when the network and nodes connected to the network are functioning normally. This rules out the case of dramatic network failure that possibly leads to disconnection of a high amount of nodes.

Every node can contact any other node, either through a direct link or by sending it's message to a neighbour, which will decide whether to send it directly or pass it on to the next node. For a node to be able to determine where to send a message it needs an understanding of the network architecture.

Because we want the network to scale by itself and without external intervention, a client-server architecture is not advised. The only way to scale client-server architectures is by providing server farms that balance requests between the different servers. Therefore such architectures require external intervention to scale e.g. by adding servers. As we do not want a network dependent on external intervention, this option is ruled out. The way to go is therefore a distributed network without a central server, as stated in our goals. This partially solves the problem of robustness as well. There is no central point that is critical to the network and that in case of failure would bring the complete network down.

The robustness of a network is not solely determined by the network architecture. However, a network architecture without a single point of failure is required. This means that disconnecting a single node from the network will not permanently disable or permanently disconnect all the other connected nodes, or even a significant subset of all connected nodes. The temporary disconnection of a small number of nodes remains possible.

As is the case with other networks, information will have to be stored somewhere on the network. As mentioned before, the network is to be distributed and therefore the data will also be distributed. Again this distributed handling of data will enhance the robustness of the network. The different kinds of data and the locations they are stored will be discussed later in chapter 3 "Data Storage".

As mentioned in part one, chapter 5 "Gnutella", a flat network architecture is not perfect, neither with regard to communication between nodes (especially searches) nor in case of network partitioning. We therefore propose a hierarchical design that will provide us with a way to organise the nodes connected to the network.

We have to deal with the hard to solve problem that there is not a single node that can be deemed robust enough to be connected all the time. **Where will we put our data? Any node can fail...**

2.3 Levels

To address the problem we have just shown, we're going to take a look at the uptime of the nodes connected to a file-sharing community:

No two nodes will have the exact same uptime, but we can still classify them into groups that share the same behaviour with regard to uptime. For example: some users will leave their pc connected to the network all the time, others will just connect to the network to download the files they need and then disconnect from the network.
The distinction between the behaviour of these two groups of users is quite clear and we will take this distinction into account when designing the network architecture. Note that more of these behaviouristic user properties will be used later on to improve the efficiency of the community.

We currently know users can be classified into different groups depending on their uptime. Similarly, we can further classify them in groups, according to the network resources they are offering to the community.

Now that we have a way to classify users by their behaviours and by properties regarding network resources, we can use this information to the advantage of the community. If we were to place some piece of information on the network that is characterised as follows:

- Very important to the network
- Accessed frequently
- Not a large volume of data.

On what type of node would we put this kind of information? Clearly this information would profit from being located on a node that has a high uptime and a high amount of bandwidth. Disk space is not important to the information, as the volume of data is low. By taking into account what preferences a piece of data has, we can map these preferences onto a group in our hierarchy and therefore onto a set of nodes. We've just used the information provided by our groups to find a suitable node for our data.

Not all behaviouristic properties we just used are important when thinking in terms of network architecture. While uptime and bandwidth are relevant when deciding on the architecture of the network, disk space is much less important. The importance of disk space increases when talking about file size. Therefore the

appreciation of properties will vary according to the position in the network that is evaluated. This discussion will be carried on later in this chapter.

We now know both uptime and bandwidth of a node are two major factors in building the network architecture. Using this, we will propose a way to organize the nodes into the network architecture using levels. Levels are just about the same thing as the groups we talked about, but we have a hierarchy between these levels instead of just having a set of groups. The levels start at the bottom with Level zero and go on virtually unlimited. Nodes that reside in the top level are the nodes that have the highest uptime and the highest amount of bandwidth. The nodes at the bottom are the ones that don't have a good uptime E.g. the nodes that connect to download a file and then disconnect again.

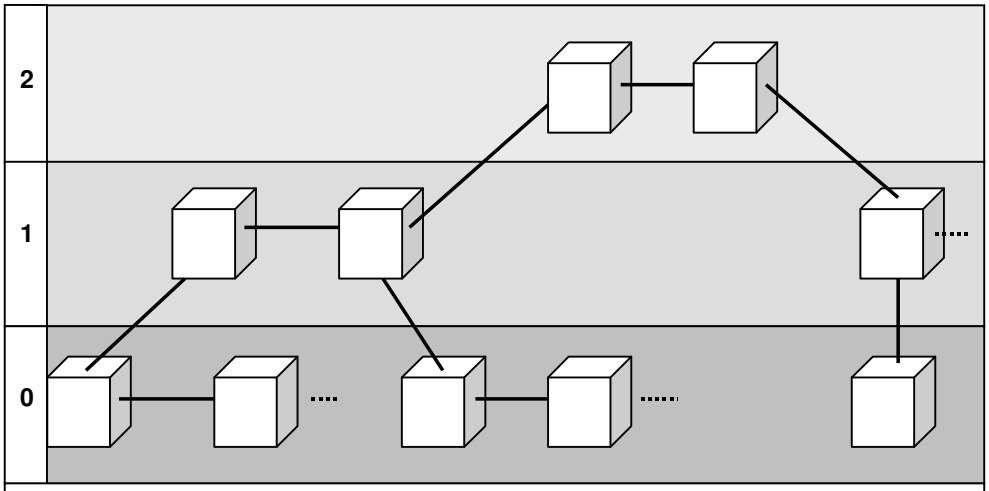


Figure 2.3.1 – *Nodes classified in Levels. The boxes depict the connected nodes and the connecting lines represent the connections maintained between them. On the left hand side, the levels are marked.*

By locating nodes with the highest uptime on top of the network, we structure the network around these stable nodes. The network can then centralize critical information on these nodes. The connections these nodes sustain will also be more stable, thus forming a stable backbone for the network.

As indicated, by using levels a structure is imposed on the connected nodes. The network structure resembles a tree. However, there is no root-node and no defined arity. Figure 2.3.1 depicts an example network in level notation. There are 3 levels in this example. The top nodes are smallest in number and the connection between them form the backbone of the network. If this connection were to be lost, the two halves of the network would not be able to communicate with each other and the network would have to go through a considerable amount of trouble to reconnect to the other half. That is why the most stable nodes are in charge of maintaining these connections. The way Shagger handles network partitioning is explained in chapter 8 "Error Recovery".

This solves the problem of where to put the data in our network. The chance of a node in a high level to disconnect is much less than the chance a node in a lower level will disconnect. Therefore critical community data will reside on higher level nodes. Of course there is still a chance that a node in a high level will disconnect from the network and therefore other precautions will have to be made so critical data is never lost. These precautions are discussed in chapter 6 "File Replication".

2.4 Building the Network

First we have a look at the way we're going to represent the network throughout this text.

Figure 2.4.1 depicts a network. The different nodes and the connections between them are depicted. The nodes are positioned according to their shared transfer speed. Transfer speed is the time taken to transfer a volume of data between two nodes. Nodes that are located further from each other share a greater transfer speed. Nodes in each other's proximity share a smaller transfer speed. Expressed in the amount of bandwidth two nodes share, this would lead to:

- Two nodes located close to each other share a higher amount of bandwidth than nodes located far from each other.

Note that the transfer speed two nodes share is influenced by a lot of factors ranging from network congestion to the actions a node is currently performing. Therefore this representation of the network is in constant change and can only be seen as the representation of the network on a short moment in time.

Now that we have a representation for the network architecture, we will transform the transfer speed notation into the level notation for a random network. This way, we will explain the Architecture of the Shagger file-sharing network.

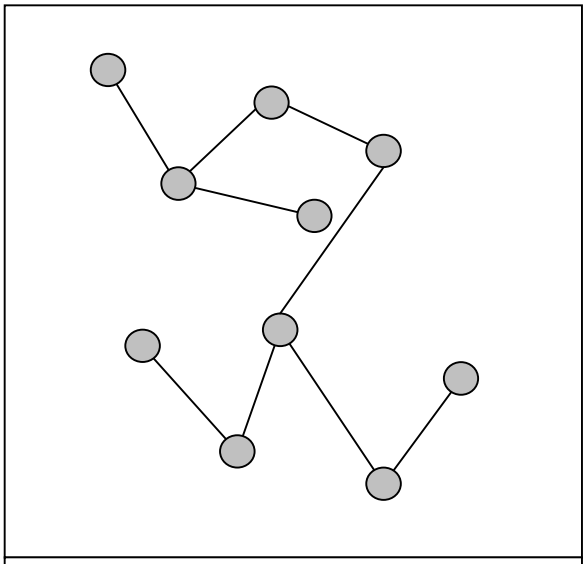


Figure 2.4.1 - *A network representation relating nodes through their shared transfer speed.*

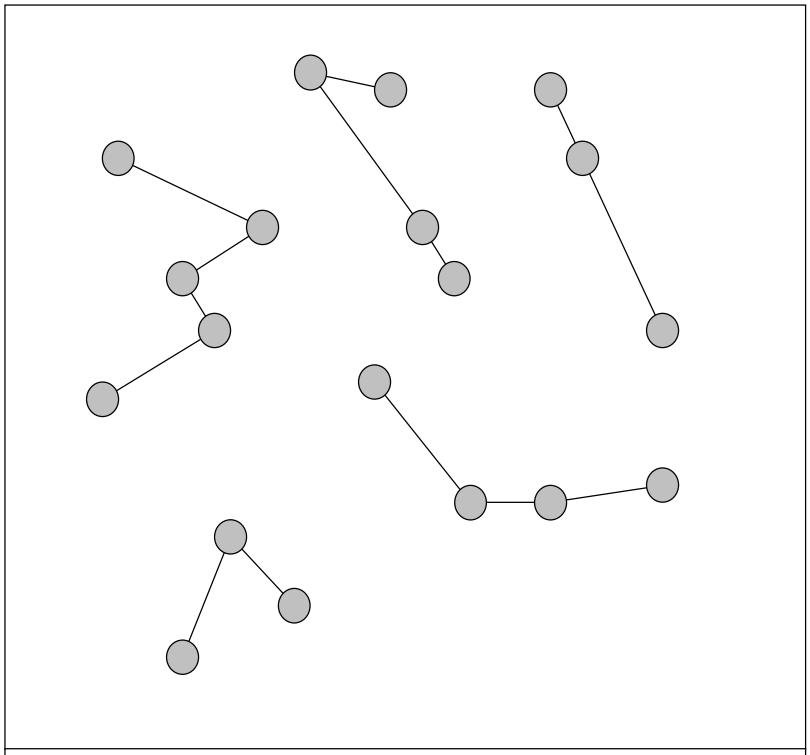
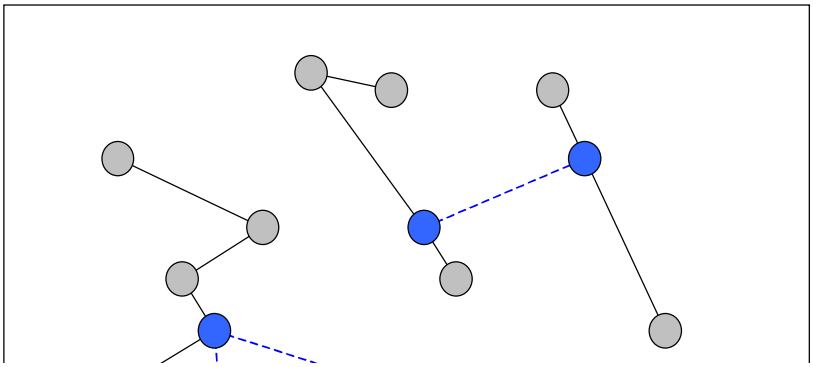


Figure 2.4.2 – *A network, again in our transfer speed notation.*

Any network can be represented by our transfer speed representation. Figure 2.4.2 depicts such a random network. Again, the different nodes with the connections running from one to another node are visible. There are five partitions that form five independent networks. If these networks are not connected, they cannot communicate.

Actually, these nodes are connected, but not on this level. Figure 2.4.2 is only showing the connections on the lowest level of the network hierarchy.



On Figure 2.4.3, the nodes and connections of level one are marked in blue. The nodes were also shown on figure 2.4.2, but then in grey. Some of the nodes that were level zero nodes, became nodes of a higher level. This is what we call promoting a node. The promoted node is the most stable from its group, and therefore gets a promotion to the next level. Although every group has a node that is the most stable of that group, the stability between promoted nodes from different groups can vary widely. Note that the criteria a node has to fulfil depends on the level a node is promoting to. We will discuss promotion of nodes more in depth after our network is converted into level-representation.

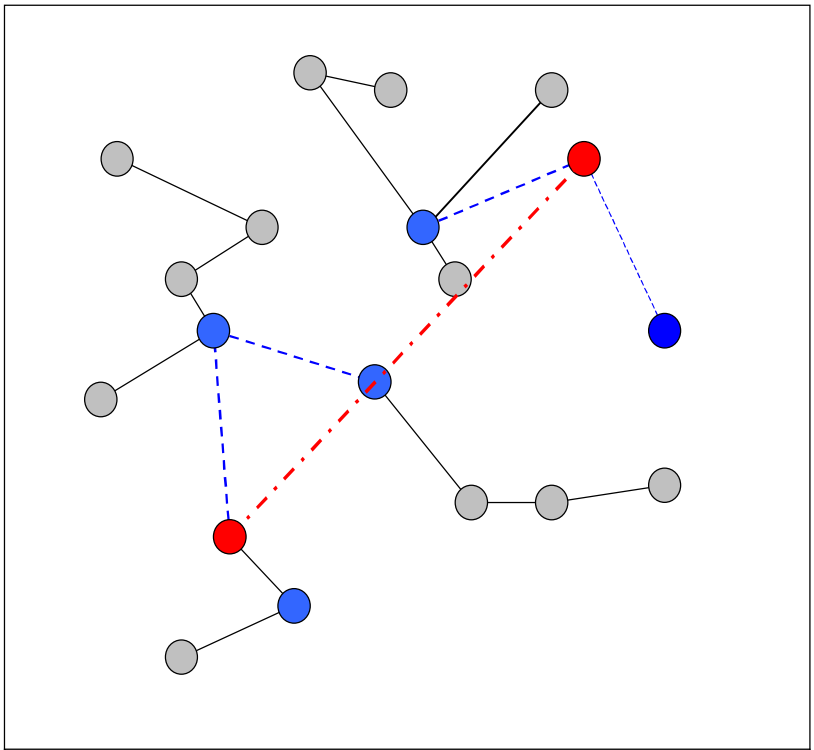


Figure 2.4.4 – *Again the same network, but now with the connections and nodes from level two indicated in red.*

displays the top-level nodes marked in red. Again these nodes have been promoted from level one to level two. But there is a difference between this step and the previous step:

In this step, together with the level one nodes that have been promoted, some nodes of level 0 have been promoted to level one. This is because linking a very stable node (in this case a level two node) to an unstable node is no good practice. Unstable nodes connecting and disconnecting would bug the stable node. We don't want unstable nodes to connect to nodes that are much more stable. Therefore we don't allow connections between nodes with a level difference of more than one level. If the level zero node is stable enough to become a level one node, it can promote to that status. Otherwise, the level zero node will be asked to disconnect and reconnect to another level one node.

It is possible that the relative weight of criteria is changed with the level a node is promoting to. For example: a node promoting to level 1 will be judged on its stability. A node promoting to level 2 will be judged on his stability and his ping time with other level 2 nodes. This is because of the change in function the node is going to fulfil on the network. For example: a level 1 node needs to be stable and have a high amount of bandwidth. A level 2 node must send messages so ping time becomes more important. Therefore PING-time is taken into account when judging a node's promotion to level 2 and not only bandwidth and stability.

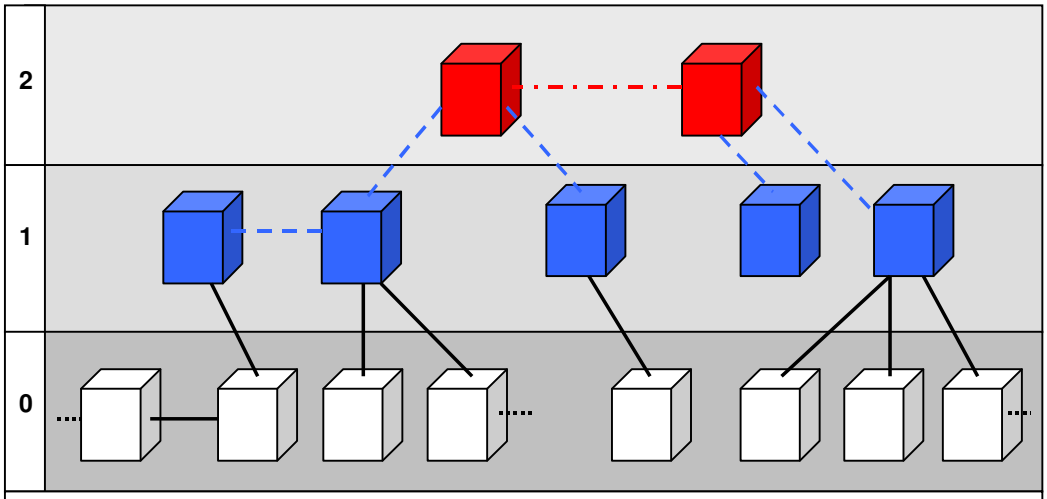


Figure 2.4.5 – *The same network as in figure 2.4.4, but then in level notation. Note that not all nodes on level 0 are displayed*

Figures 2.4.4 and 2.4.5 actually represent the same network. Figure 2.4.4 is a representation of the network in transfer speed notation and Figure 2.4.5 is the same network in level notation. Nodes and connections where given the same colours on both figures to make the similarity obvious.

While we now know how the Shagger network looks, we don't yet know how nodes get selected for promotion and how these nodes get promoted. The best way to explain what happens in case of a promotion is to show the different situations that can trigger a promotion:

- The connection of a new node to the network.
- The reconnection of a node to the network.
- The disconnection of a node.

All three cases will be discussed here, but first we'll have to define an extra notion that will help us better understand the different situations.

2.4.1 Action Radius

The action radius of a node can best be described as the part of the network it covers. The amount of the network it covers depends on the bandwidth a machine has at its disposal. Depending on this bandwidth, a threshold is calculated. When calculating whether a node lies within the action radius of another node, the transfer speed to that node is tested. If the transfer speed lies below the threshold, the node lies within the action radius. Nodes from level zero don't have an action radius. Only nodes starting from level one have an action radius.

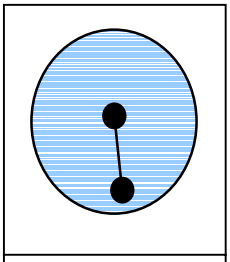


Figure 2.4.1.1 – *An action radius*

An action radius is depicted, as in figure 2.4.1.1, by using a blue circle. In the same picture, two nodes lie inside the circle and are connected to each other. The node in the centre is the owner of the action radius. The fact that the other node lies within the circle means that this node is within the action radius of the centre node.

On figure 2.4.1.2 two action radii are depicted. The two nodes are the owners of the action radii. A node can never have best transfer speed to two nodes. Therefore the action radii can never overlap. The line that is perpendicular to the connection between the two nodes indicates the border of the two action radii.

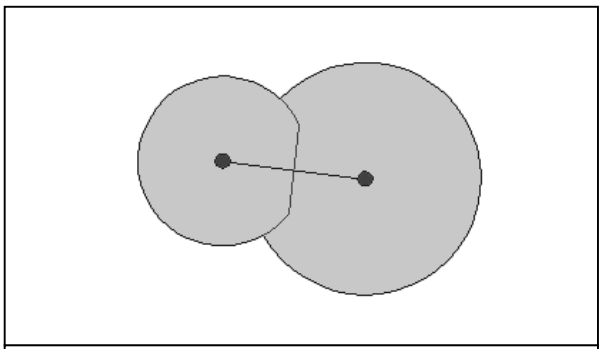


Figure 2.4.1.2 – *overlapping action radii.*

When we look at an example network and display the action radius for every first level node, we get the situation as depicted in figure 2.4.1.3.

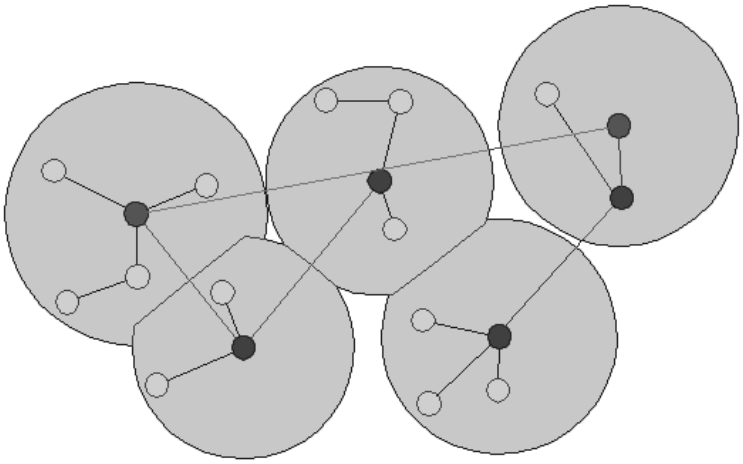


Figure 2.4.1.3 – *An example network with the action radius of every first level node visible.*

Now that we know what an action radius is, we can start using it to explain the different cases that trigger a node to level. The action radius is used to determine the connection point for a connecting node.

2.4.2 Connection of a New Node.

The problem with a new node that wants to connect to the network is finding the place in the network where it has to connect to. The goal is to locate nodes that share high bandwidths close to each other in the network hierarchy.

The first task for the network is to find out where the node is located. When it knows where the node is located, it can find other nodes that share the highest bandwidth with the new node. Logically the new node would then be connected to these nodes. Recall that in our transfer speed notation, the nodes that share high bandwidths are located close to each other.

Note that to initially connect to the network a new node can connect to any node on to the network. The new node will then be redirected to one of the top nodes.

Determining the best connection partner for a new node:

To determine the best connection partner for the new node, the algorithm starts at the top of the hierarchy. The top nodes determine in parallel (all nodes from that level) which of them has the best transfer speed to the new node. If there is a top node that has a better transfer speed than others, this node recursively calls the same routine on its children to determine which of them is closest to the connecting node. In terms of action radii this means that the action radius to which the connecting node belongs is determined. The controller of this action radius shares the highest amount of bandwidth with the new node.

If the new node lies beyond the action radius of every level one node, the network does not yet cover the location where the node resides. This means the new node will have to be connected to the closest level one node for now. The node to connect to has to be a level one node as connections are only allowed between nodes of 1 level higher or lower than the level of the connecting node. In this case the connecting node is a level zero node and therefore can only connect to a level 1 node.

The connection of a new node to the network can trigger other nodes to level. Levelling of a node only takes place if that node conforms to all the requirements to promote to the next level. The requirements to promote to level 1 include:

- Stability and sufficient uptime
- Sufficient bandwidth

Level:	**0**	**1**	**2**	**3**	**4**	**5**	**6**
Stability (uptime)	0	1	2	3	4	5	6
Disk Space	0	1	1	1	1	1	1
Amount of bandwidth	0	2	1	1	1	1	1
Ping-time to other nodes of the same level	0	1	2	3	3	3	3

Table 2.4.2.1- *The Relative weight of properties in relation to levels.*

taken into account when levelling a node. There are 7 different levels listed in the figure. However a virtually unlimited number of levels is possible. The only property that changes importance from level 4 onward is the stability of a node.

The next example will explain levelling triggered by a new node connecting to the network:

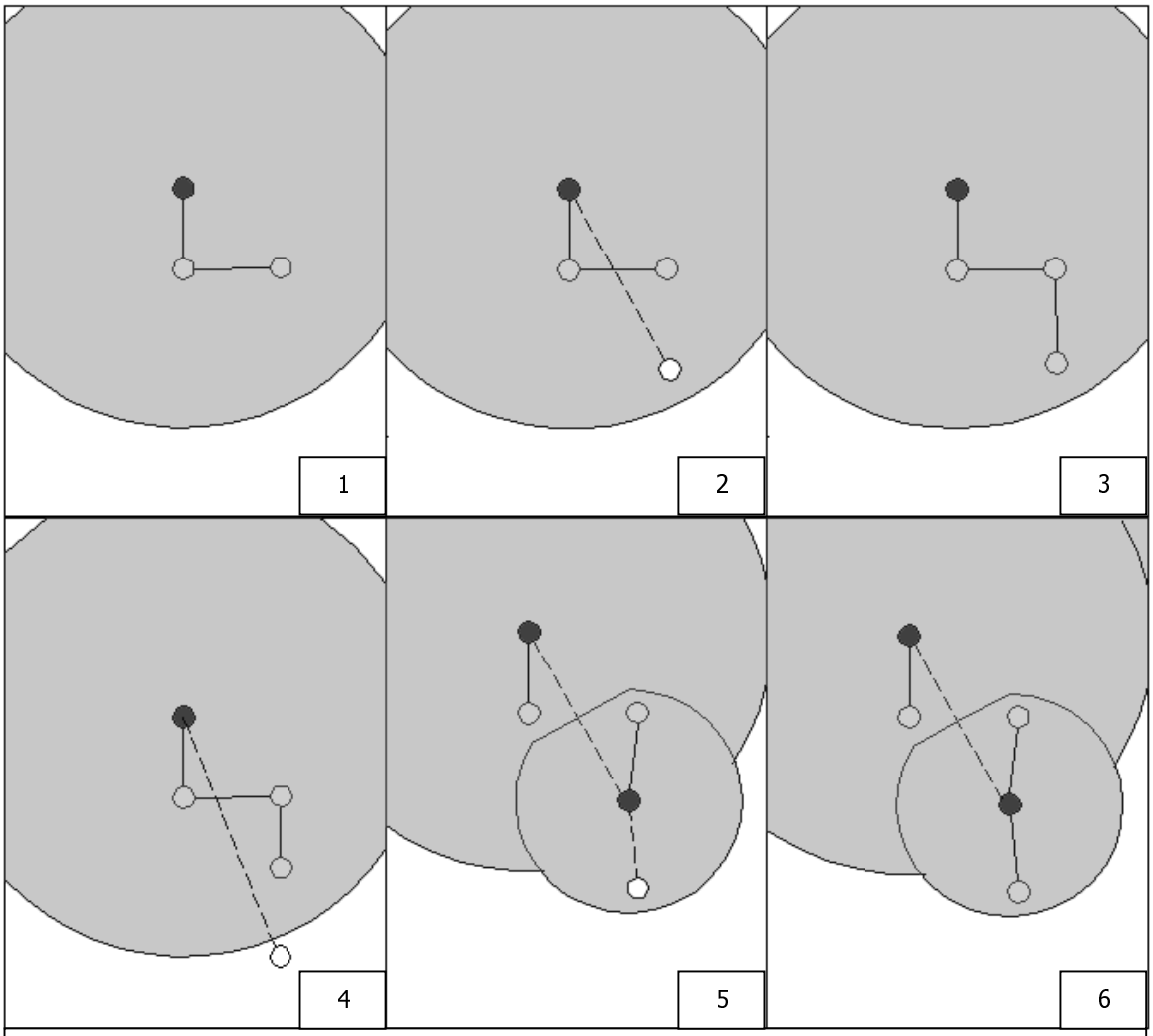


Figure 2.4.2.2 – *Connection sequence of a new node to the network. In frame 5, a node has been triggered to level.*

The sequence depicted in figure 2.4.2.2 displays the changes the network goes through when a new node connects.

The explanation of what happens in every frame:

1. There is one level one node and all the other nodes are level zero nodes.
2. A new node wants to connect to the network. The new node is displayed in white, as it has no level assigned to it yet. It is connected to the top node and will be assigned a connection point in the network by using the method we saw earlier.
3. The new node has been given a connection point and has connected to that node. It is now displayed in grey as it resides in level zero.
4. Again a new node wants to connect to the network. In this case however the node is located outside the action radius of the top node. This triggers a reorganisation of the network to accommodate the new node.

5. The reorganisation is complete and a level zero node has been promoted to level one. An action radius was added and one connection was dropped. The connecting node has connected to the top node in whose action radius it lies.
6. The new node's connection point was found

This leveling technique is used when new nodes connect to the network not only to make sure nodes that share a high amount of bandwidth are clustered together, but also to keep the network balanced. This is better explained using the level notation introduced earlier.

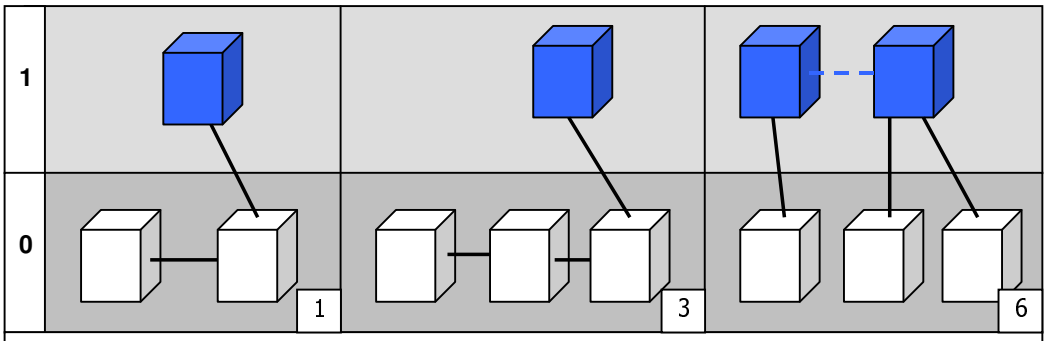


Figure 2.4.2.3 – *The sequence of figure 2.4.2.2, but shown here in level notation.*

When looking at figure 2.4.2.3 it is clear that not levelling a node in frame 4 would lead to a long chain of connected level zero nodes. Messages sent by the last node in the chain would have to travel through all the other level zero nodes before the message would reach the level one node. After the reorganisation, the nodes can reach a top-level node much faster. In this case even through a direct link. When this same strategy is applied to the entire network, there is a good balance between the level of a node and the number of children of that node. This network layout has many advantages regarding message passing as it minimizes the number of nodes a message traverses.

2.4.3 Reconnecting a Node

Reconnecting a node to the network is similar to connecting a new node. Shagger stores user data and the level of a node. We therefore know what level a node belonged to before it disconnected. Where the user data is stored is described in chapter 3 "Data Storage".

Determining the best connection partner for a reconnecting node:

It is possible that the network changed drastically while the node was disconnected so using the node's old connection point is not an option. We use the same approach as we used when locating a connection point for a new node: the reconnecting node is first connected to a top-level node and forwarded downwards throughout the hierarchy until it reaches the level it belonged to. The branch that is descended is selected in the same way a new node's branch is chosen: the top nodes determine their connection speed with the reconnecting node. The top-node that shares the highest transfer speed with the reconnecting node forwards the node to his children, etc.

A new node cannot be promoted to a level higher than zero. However, a node that has already reached a higher level before it disconnected will be reconnected to other nodes of that level, if possible.

When a node has descended the network hierarchy and it has found the level it belongs to the network has to be restructured to accommodate the reconnecting node. This restructuring of the network is independent of the level of the reconnecting node. Figure 2.4.3.1 displays and example network before network restructuring. The connecting node is of level n and the node will therefore connect to that level.

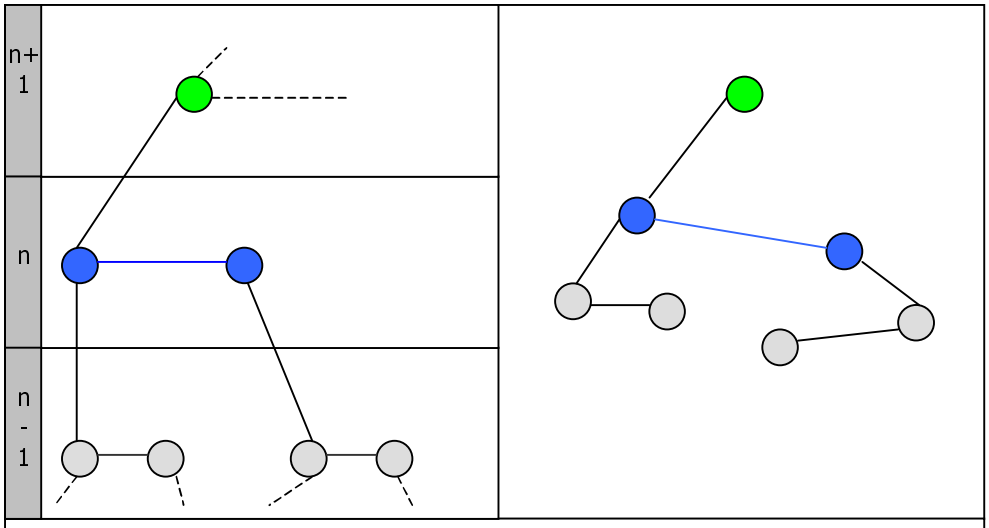


Figure 2.4.3.1 – The same network in level notation and in transfer speed notation.

Figure 2.4.3.2 displays the action radius of every level n node and the action radius of the connecting node. The connecting node will connect to one of the level n nodes. The two nodes that lie in the action radius of the connecting node will be disconnected from the network and reconnected to the connecting node.

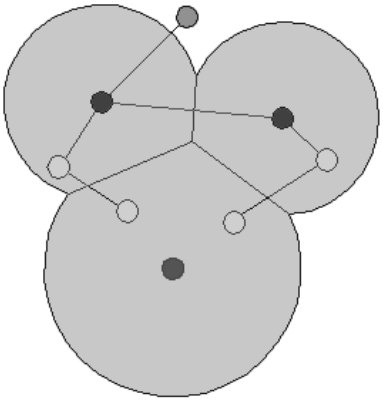


Figure 2.4.3.2 – Again the same network with the action radius shown for every level n node and the connecting node.

Figure 2.4.3.3 shows the network after it has been rearranged.

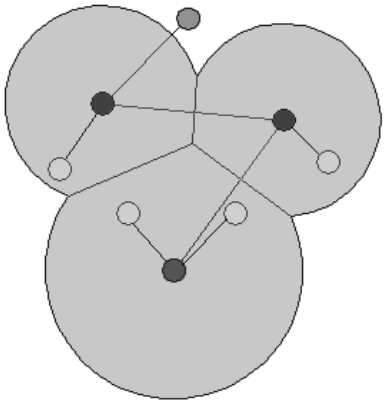


Figure 2.4.3.3 – Again the same network after reorganisation.

In this case there were nodes in the action radius of the connecting node. However, when there are no nodes in the action radii of the connecting node, the node will not be connected to other level n nodes. Instead the connecting node will be connected to nodes of one level lower. This degradation is the penalty the connecting node incurred for disconnecting from the network. The penalty is only applied if there are other nodes that have taken over the action radius of the connecting node. Figure 2.4.3.4 shows the case in which the connecting node will incur a penalty for disconnecting.

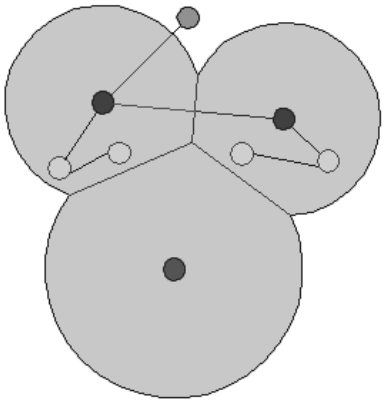


Figure 2.4.3.4 – Another network where no nodes are located in the action radius of the connecting node.

The connecting node (coloured red) will be connected to nodes of level n-1, as it would be unfair to let a reconnecting cause a connected node to be degraded.

The penalty for disconnecting from the network is therefore not always applied. Only if the reconnecting node is miss able to the network, the node will be degraded.

2.4.4 Post-(re)connection Tasks

After a node was assigned a connection point in the network it must perform certain actions.

On the (re)connection of a level zero node its shared files are registered with the parent level one node. The exact explanation of how files are registered and what actions are performed to do so, are explained in chapter 5.2 "Identity search".

Reconnection of nodes of a higher level than zero is more complex:
The actions a reconnecting node has to perform are different for level one nodes and nodes of a level higher than level one. First we will discuss the connection of nodes with a level higher than level 1.

Reconnecting nodes that have been connected to the network must perform the following actions:

- Build a shared files list.
- Update their user data.
- Clear their statistical data.

What these kinds of data represent and what they are used for is explained in chapter 3 "Data Storage".

Level one nodes have to perform the same actions as a higher-level node and some more:

- Gather statistical information of "his" nodes.
- Index shared files on connected level zero nodes.

What we mean by "his" nodes is explained in chapter 6 "File Replication".
Nodes of level one and higher are now connected to the Shagger network.

2.4.5 Disconnection of a Node

The case where nodes connect to the network has been explained in the previous part. Here we will concentrate on nodes that are disconnecting from the network. This can happen in two ways:

- A node can make a clean disconnect. By a clean disconnect we mean that the disconnecting node first makes sure it's responsibilities are taken over by other nodes in the network and only then disconnects from the network. Therefore no critical data is lost.
- A node can stop functioning and ignore all requests from other nodes in the network. This is called a dirty disconnect. In this case the disconnecting node failed to appoint nodes to take over his tasks in the network. Therefore critical data can be lost.

When a node disconnects from the network all information stored in that node is lost to the network. During a clean disconnect critical data must be transferred to another node before disconnecting. The amount of information is dependant on the level the disconnecting node has in the network. In for example a level zero node, no critical information is kept. However, a top node stores a lot of critical information. In case of a dirty disconnect the information stored in the disconnected node must be regenerated. In some cases this is impossible and information will be lost to the network. What will happen to the network in case of a dirty disconnect will be explained in chapter 8 "Error Recovery". Here we will concentrate on what happens in case of a clean disconnect.

What we again strive for is a balanced network. When talking about new nodes that are connecting to the network, we used leveling to keep the network balanced. Here we will do so again, but instead of promoting nodes, we are going to degrade the disconnecting node. Once the node reaches level zero, the node can be disconnected from the network without losing critical information. As nodes located in the top levels will disconnect far less than nodes at the bottom of the hierarchy, the disconnection of lower level nodes is easier than the disconnection of high-level nodes.

Figure 2.4.5.1 Displays an example network prior to the disconnection of node labeled 1.

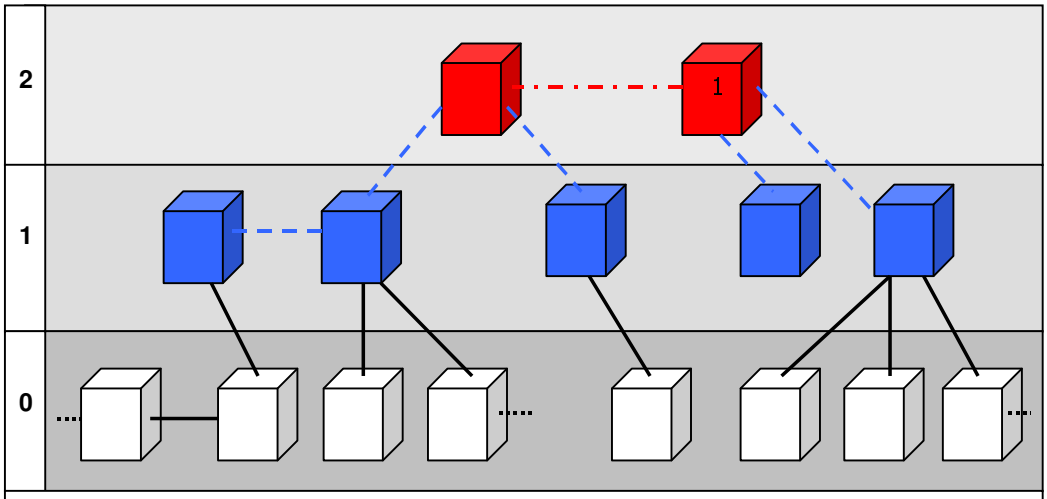


Figure 2.4.5.1 – *Example network where level 2 node (labelled 1) will disconnect from the network. All nodes and connections are shown prior to the disconnection.*

The node labeled 1 is a level 2 node. To disconnect a level 2 node we will have to degrade it 2 levels. Therefore the node (labeled 1) will become a level zero node. Then the node can be disconnected from the network.

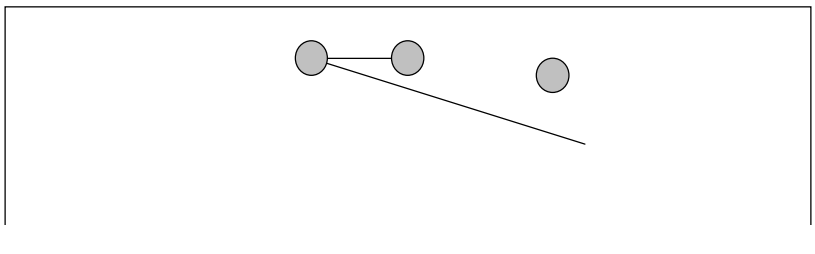
By simply degrading the level two node critical information will still be lost. Therefore children of the level two node will take over a part of the node's critical information. In chapter 3 we will introduce the different types of data on the network and evaluate if they represent critical information. Therefore in this chapter we only state that user data is critical information. All the user data from the level two node is therefore first transferred to its children. If the disconnecting node is a level one node the user data is transferred to the parent and not to one of his children. Children of level one nodes are level zero nodes. These nodes are typically unstable, have low bandwidths, etc. Therefore, level zero nodes will not be trusted with critical information.

Note that transferring user data will not take long, as user data is typically not larger than a few kilobytes.

When all user data has been transferred to a child or parent the level two node can be degraded. Before it can be degraded it has to find a node to take his place in the hierarchy. One of his children will be the first choice.

Therefore, if one of the children is ready to level, it will do so. If there is none of his children that is ready to level, we will reconnect all children (together with their children) of the disconnecting node to the network. The reconnection will not be done through the top-nodes of the network but through the parent node of the disconnecting node. If the disconnecting node has no parent node, the top nodes of the network will be used to reconnect.

The disconnecting node will at this time no longer be taken into account when leveling or reconnecting any node. However, it will remain connected to the network. Figure 2.4.5.2 displays the example network in transfer speed notation with on of the children of the disconnecting node promoted to level 2.



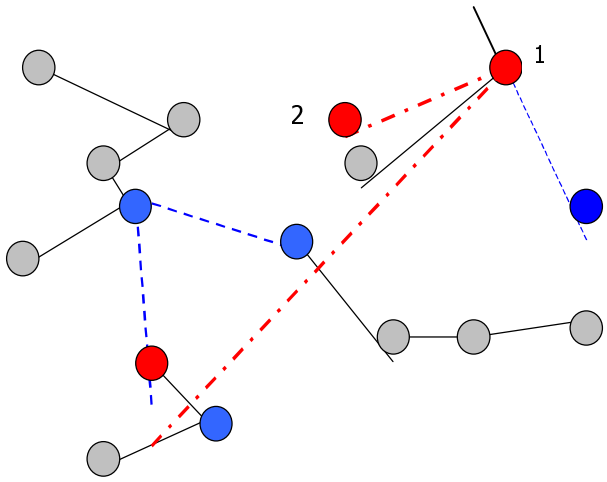


Figure 2.4.5.3 – *The example network in transfer speed notation. The node labelled 1 is the disconnecting node.*

Figure 2.4.5.3 displays the level 1 node that has been promoted to a level two node in red (2), the level two connection (dotted red) is still running from one level two node to the disconnecting node (1). This connection will have to be dropped and reconnected to the promoted node (2). It is also clear that all level zero nodes that where connected to the promoted node (2) are now connected to the disconnecting node (1).

Next the remaining children of the disconnecting node are reconnected to the promoted node (2). This is shown in figure 2.4.5.4 together with the disconnecting node's new level. The disconnecting node is now a level 1 node. The disconnecting node will now be degraded to a level zero node. Therefore the same procedure will be repeated as when degrading a level two node: Critical data is copied to the parent of the disconnecting node. Note that in this case there is no more critical information located on the node. We deleted all the critical information from the node when de degraded the node to level 1. Next a replacement node is chosen from the disconnecting node's children. The chosen child node will therefore be promoted from level zero to level one. If there is no node ready to level the children will be disconnected and reconnected to the network as we stated before when the disconnecting node was degrading from level 2 to level 1. In this example there is a node ready to level.

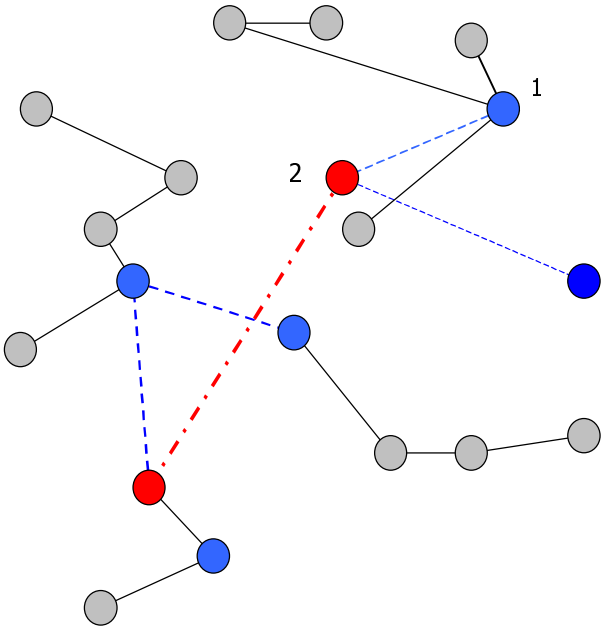


Figure 2.4.5.4 – *The disconnecting node (1) has been degraded to level one.*

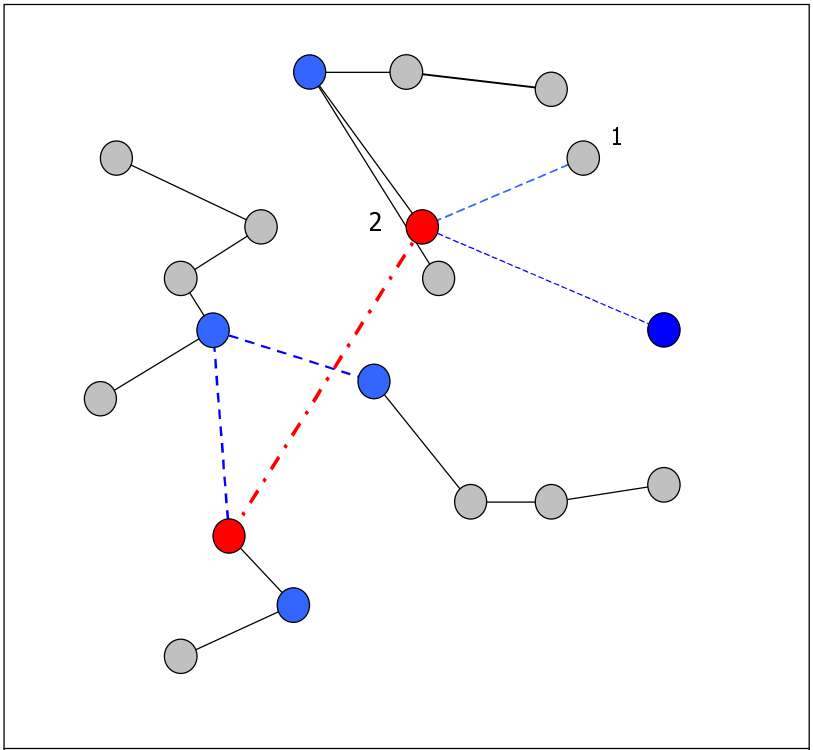


Figure 2.4.5.5 – *The disconnecting node (1) has been degraded to level zero.*

Figure 2.4.5.5 represents the network after the disconnecting node reached level zero. The disconnecting node can now disconnect from the network without loss of critical information.

2.5 Conclusion

Throughout this chapter we have introduce different techniques e.g. Levels, action radius, promotion and degradation. We will summarise why we used those techniques:

- All of these techniques make sure the network stays balanced.
- By letting the most stable nodes form the backbone of the network we improve the robustness of the network. We can identify the most stable nodes of the network by looking at the level of a node.
- Degradation makes sure the network loses no critical information.
- By using 2 action radii we can determine the nodes that share the best transfer speed.

Files are not the only kind of data stored on the Shagger file-sharing network. Data needed to manage the community and the users is also stored on the network. This chapter deals with identifying the location where data is stored on the Shagger network and what kind of data is stored.

As we indicated in chapter 2 "Shagger network architecture", there is a variation in the stability of the nodes in the network. We classified the nodes into different levels according to their stability. This way we determined where what kind of data could best be stored. For example: storing critical information on a node of level zero is not such a good idea. Critical information can best be stored in the higher-level nodes.

Even nodes located in the highest level are not fail safe. Therefore, even these nodes can disconnect from the network without prior notice. Although the chance of this happening is much smaller than the chance of a level zero node disconnecting from the network this case still has to be taken into account.

By means of replication, data can be stored on different locations. This minimises the chance of losing data. Replication and problems associated with using this technique will be discussed in the chapter 6: "File Replication".

We know we have to store critical data in the high level nodes. This leaves us with the task of identifying the critical data. Before we can do that, we need to examine all kinds of data present on our network.

- File Chunks
- Shared Files List
- Index List
- User Data
- Statistical Data

We will explain all of these data types in more detail. In chapter 3.6 we will discuss the locations where these different kinds of data are stored.

3.1 File Chunks

File chunks are parts of a file. Figure 3.1.1 displays a file divided into file chunks.

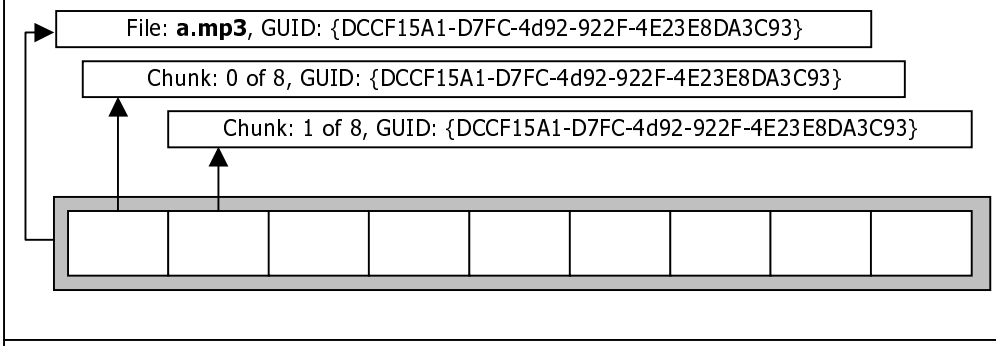


Figure 3.1.1 – *A file divided into file chunks. Every file is uniquely identified by a GUID and so is every chunk by its chunk number and the GUID of the file.*

As indicated on Figure 3.1.1, a Globally Unique Identifier (GUID) [13] uniquely identifies every file. This GUID is a generated value that is guaranteed to be globally unique. We assign the same GUID to every chunk of a file together with a chunk number. This again uniquely identifies every chunk of a file. The additional data that is stored for a file and a file-chunk is not displayed. For every file on the network we store its filename, size, checksum, and its type. For every chunk we store, besides the GUID and chunk number, the checksum for that chunk and all information associated with the file it belongs to.

All chunks are of equal size. This size these chunks should have is however open to statistical research. Therefore we will not state a value in this thesis.

3.2 Shared Files List

Every node has a shared file list. This list indicates what files the node shares. The list is built up out of filenames and GUIDs that identify those files. On figure 3.2.1 an example of such a shared file list is given for an example node.

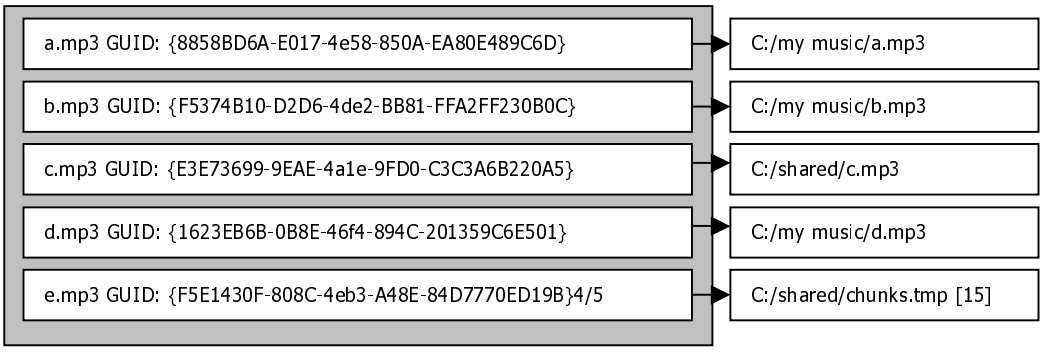


Figure 3.2.1 – *A Shared file list for a random node maps file names and GUIDs onto locations on the hard drive.*

On Figure 3.2.1 the mapping between the shared files list and the logical location of the files is displayed. A shared files list contains a mapping for every file and chunk a node shares. When a chunk is mapped in the list, an index in a file is added to the location. Through this index the position in the file can be determined where the chunk is stored.

When building this list the node has to check with the rest of the network if the files or chunks are available or not. If a file seems to be new to the network, a new GUID is assigned to the file. If the file is present on the network, the GUID of that instance is used to identify the new file. How the file is located will be explained later in chapter 5.2 "Identity Search".

3.3 Index List

The Index list maps file names and GUIDs to level zero nodes where files or chunks are shared. Figure 3.3.1 displays this mapping.

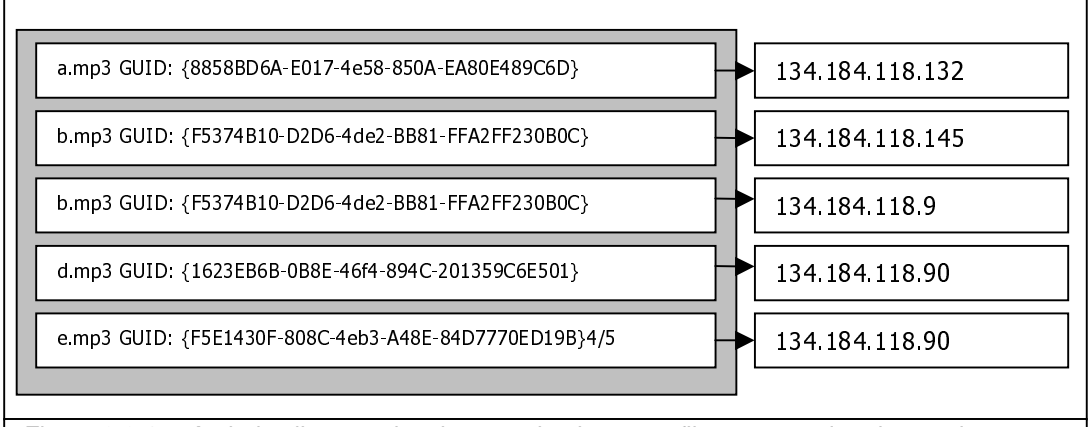


Figure 3.3.1 – *An index list contains the mapping between filenames and nodes on the network.*

Figure 3.3.1 shows that for every file shared in one of the nodes that this list indexes, there is a mapping to an IP-address. Every level one node maintains such an index list for all the level zero nodes that are connected to it. Figure 3.3.2 displays this technique in level notation.

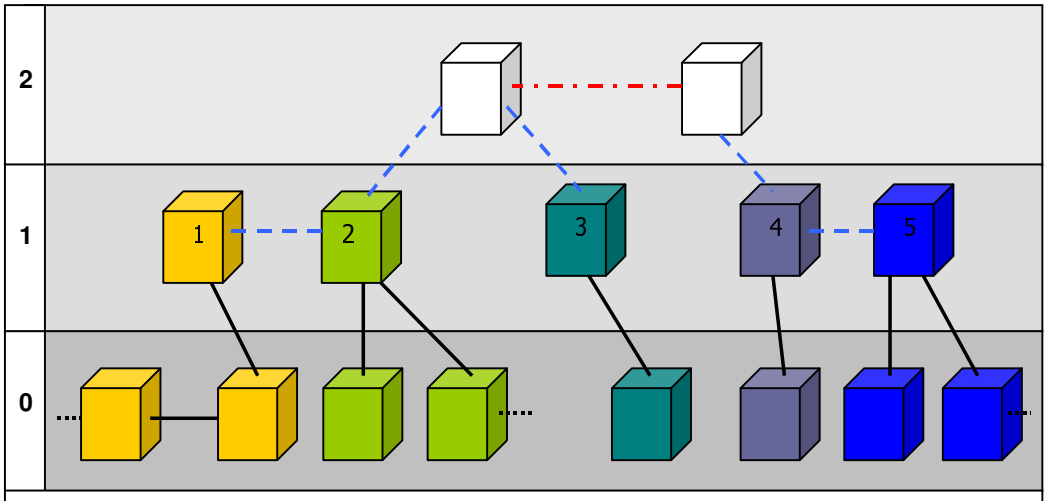


Figure 3.3.2 – *The yellow level zero nodes are indexed by the yellow level one node, the green level zero nodes are indexed by the green level one node and so on. The level two nodes are only displayed for completeness.*

In figure 3.3.2 the yellow level one node (marked 1) indexes all the yellow level zero nodes. This means that it maintains an index list that maps the files and chinks shared to the IP-address of the node that shares the file or chunk. The result of this is that all level one nodes can anser to search queries like "Is the file identified by GUID {3D1BB393-7A08-4c41-A152-E89A808D0A91} shared by one of the level zero nodes connected to you". Level zero nodes are unstable and only have a small amount of bandwidth. Therefore the amount of queries level zero nodes have to deal with is kept to a minimum. This is done by letting a level 1 node index their shared files. The level one nodes can therefore answer any queries meant for his level zero nodes. The index list is updated whenever one of its level zero nodes connects, disconnects or removes or adds shared files.

Only nodes of level zero are indexed by building an index list. Nodes of level 1 and up are not indexed in any way. Attempts to index all nodes on the network would lead to update problems. For example: as we are dealing with a distributed network, data is not centralised. Therefore index lists on all nodes would be located all over the network. If one of these copies is not updated, incorrect information could lead to errorous behaviour by nodes.

3.4 User Data

The Shagger network has to keep data about different users. This is because of the way the network structure depends on the behaviour of users. E.g. when a user's node has reached level 4, meaning that the node is a stable one, it will be reconnected to level 4 if possible. To do this we need information about a user, that remains on the network and that is retrieved when that user connects to the network. If we would store no information on users, we would be unable to study the behaviour of users.

Once a user's node deserved to be promoted to a high level and by this has proven its stability, the node will be deemed stable until it has proven otherwise. By disconnecting once from the network in an orderly fashion the node might be degraded one level, but certainly not more. To be able to determine the stability of a node connecting to the network, so that it can be connected to the right level, a record has to be kept of the node.

This record is stored in files on the network. These files meet the same requirements as normal files: They are identified by a GUID and have the same properties a normal file has. However, user data cannot be divided into chunks.

The user data file will contain the user's name, password, GUID and the level that was reached by his node. In case of a new user connecting to the network, we do not retrieve user information.

3.5 Statistical Data

Statistical data is stored on every node of level one and upward. This data represents downloads and searches that have been performed by users. Statistical data is deleted and rebuilt after a certain amount of searches. Every node keeps this information on its own searches and downloads. However, a level one node also keeps statistical information on other nodes. This information is used in the replication strategy we will introduce in chapter 6 "File replication".

3.6 Data Location

The only place in the network where we can store data is on the nodes making up the network. We will identify the kind of nodes we are going to store the data on. Data locations are identified by levels. When we state we can store data on a level, we can store this data on any node of that level.

- File Chunks are stored on the nodes that share the file to which the chunk belongs. As every node on the network can share files, any node can have file chunks. File chunks are therefore located all over the network on any node of any level.
- A list of Shared files is stored on every node that shares files. We can make the assumption that every user can share at least one file. Therefore files can be shared in every node throughout the network. Therefore any node on the network has a shared file list.
- Index lists are only located on level one nodes. As explained earlier the level one nodes index the files shared on the level zero nodes connected to that level one node. The most logical location to store these index files is therefore on the level one nodes.
- User data must be available to every node on the network at any time. This data is very critical to the functioning of the network. If this information is lost the user's Level is lost and he will have to start off at level zero again. Therefore we locate this data in the highest levels.
- Service data is located where the service is provided to the network. Typically this is on one node only. Different nodes can provide the same service, but typically the services are not interconnected.
- Statistical data is stored on every node from level one upward. Cumulative statistical data is only stored on level one nodes.

3.7 Conclusion

We showed what kinds of data are present on the Shagger network and where they are located. We identified the critical data as being the user data. All the other data available on the network is treated as data that can be regenerated. How regeneration occurs for the different kinds of data is explained in chapter 8 "Error Recovery".

The following chapters will use the different kinds of data we discussed to implement the standard services: user management, searching, replication and downloading.

In part I, chapter 1.3 "Goals" we stated that we are convinced of the added value of providing a more elaborate social context for interactions conducted in file-sharing communities. This chapter will discuss the technology used to implement the user management, but also explain what we want to gain by offering a better social context for users of the community.

Current file sharing communities do not offer any real user management. They only use user information to identify a user. In the Shagger network this information is not only used to identify a user, but also to monitor the behaviour of that user. With the information gathered on the behaviour of a user, the class that best matches a user is determined. In earlier chapters we introduced levels in our network and used them to illustrate different cases that trigger a node to level. These levels represent the different behaviouristic classes a user can belong to. For example: a node of a user that belongs to the level zero behaviouristic class is characterised by instability, low bandwidth and a low uptime. The partitioning of users in different levels is therefore a way to classify users to determine the best position in the network for their node.

We use this classification system In the same way to determine the kind of files a user is interested in. When the interests of a user are determined, the system can anticipate user's actions. This reduces the time needed to fulfil requests made by users. For example: if a user downloads a lot of movie trailers, the network can present a list of trailers that were recently added to the network. This and more of these services are explained in more detail in chapter 9 "Future Work". The services discussed in chapter 9 are implemented on top of the standard services.

This kind of user management could be easily implemented in a client-server architecture, however in a completely distributed architecture implementing this service is not a straightforward operation.

We discussed user data in the previous chapter. User data contains the name, password, GUID and the level of a user. We now describe the way this data is used to provide a distributed user management.

The information contained in the user data is stored on the network for every user with a node of level one and upward that connects to the network. Users with level zero nodes tend to frequently connect and disconnect from the community. Keeping information on these users would only stress the community resources without having any use to the community.

User data is stored in the same way a file is stored on the community network. As this data is critical to the operation of the community it is frequently replicated and therefore stored in many different nodes at a time. The file is identified by a GUID, like any other file in the community. When a node and therefore a user changes any of his properties e.g. the level it belongs to, this information has to be updated. If we would search the network for the GUID identifying the file and then replace every one of these files, update anomalies would be possible. For example: if a user data file is stored on a node that disconnects from the network, this file is not found when the network is searched for the GUID identifying the file. This way the file is not updated. When this node reconnects to the network there will be two different versions of the user data file available on the network. This could lead to unexpected behaviour when the owner of the user data file connects to the network.

We solve this update anomaly by changing the GUID of a user every time one of the user's properties, stored in the file, changes [14]. For example: If a user already has a user data file stored on the network and the level of the user changes, a new file is generated with a new GUID uniquely identifying it. The same GUID is also stored on the user's node. This way when the user reconnects to the network his node issues a search query for the new GUID identifying the new user data file. The network returns the location of the user's data file and the authentication procedure can start.

If nodes would add user data to the network every time they change level or reconnect, the network would fill up with old user data. Therefore, every node is responsible of cleaning up his old user data. There can be at most 3 generations of one node's user data on the network. The user data older then three generations is deleted from the network. We keep three generations of user data to perform error recovery in case last generation user data is lost. This error recovery technique for user data is further discussed in chapter 8.2 "Loss of User Data".

The authentication procedure is limited to matching the username and password supplied by the user to the node with the username and password contained in the user data file. As stated, user data files are located on the network. This means this information is located on the hard disks of multiple users. Note that to protect the user's network identity the information contained in a user data file is encrypted. What happens when user data files are lost is described in chapter 8 "Error Recovery".

We discussed the way the authentication procedure is performed on the Shagger network and we know users are classified into different behaviouristic classes called levels. When the network has authenticated the node, the node can take its rightful place in the network architecture. This is achieved through the algorithm presented in chapter 2.4.3 "Reconnecting a node".

4.1 Influencing the Behaviour of a User

We will now introduce a user management technique that will encourage the community's users to adapt their behaviour to the network's needs. In current systems a user of a file-sharing community does not care about other users that want to download his files. Even if other users download his files, the amount of bandwidth he can use to download files will decrease. Some file-sharing communities allow their users to specify the maximum amount of his bandwidth that can be used to upload files to other users. By taking the maximum upload speed into account when determining the maximum download speed the community can force users to let other community members download files from his machine at a reasonable speed. For example: if a user limits his upload to 5KB/s the community client automatically limits his download speed to 20KB/s. Although this value is higher than the upload speed downloading at this speed is considered slow. If that user knows that if he raises the upload limit to 10KB/s he can download at an unlimited speed he will prefer this configuration. On the other hand no user will voluntarily raise the upload limit above 10KB/s, as he gains no more bandwidth or any other advantage. Therefore although this technique almost guarantees that a user will not limit his upload speed to a value lower than 10KB/s it also leads to a considerable amount of the users setting their upload speed to the 10KB/s boundary value.

The technique we introduce will not punish users that do not share with the community. Instead, users that display "good" behaviour are rewarded with

advantages over other users. One of the behaviours that is appreciated on the Shagger network is the improvement of a node's uptime. When a node improves its uptime (and meets all the other requirements) it can be promoted in the network architecture. Therefore the rewards or abilities users receive can be associated with the level they belong to.

A reward a user receives is a one-time gesture from the network that the user can use to its advantage.

An ability on the other hand is an advantage a user gains for as long as he remains at least in the same level.

For example: we will show in chapter 5 "Searching" that a network wide search is a very costly operation regarding network resources. Level zero users will not be granted this advantage. If one of these users performs well, but cannot be promoted to a level one node because of his low bandwidth, he is rewarded by e.g. granting him a number of network-wide searches. For a short time users can therefore gain an advantage over other users through rewards.

We again use the same example, but this time the well performing node can be promoted to the next level. By promoting the node to level one it gains an ability. This ability could e.g. be network wide searches. The node can perform these network wide searches for as long as he remains a level one node. This grants the node a constant advantage over lower level nodes because of his improvement in performance. The higher a node climbs in the hierarchy the more abilities he will be rewarded with.

4.2 Ability per Level

The Shagger network will accommodate more levels as the number of nodes increase. For every level that can be reached a new ability has to be provided. As the number of levels are virtually unlimited we would have to provide an unlimited amount of abilities, which is impossible. Therefore, we will only provide abilities for the number of levels that are reachable. If the network would grow past these boundaries, the number of abilities can easily be increased, as they are built into the community application. Therefore an update of the client application would suffice to add extra abilities to the network.

The mapping between abilities and levels depends on the user's appreciation of the different abilities. To determine the appreciation of the different abilities, a study among users of file-sharing communities is needed. Therefore we will not propose a mapping in this thesis, as we have no real view on the appreciation of the different abilities.

4.3 Conclusion

In this chapter we introduced a user management technique that can influence the behaviour of users. The users are rewarded for "good" behaviour and therefore influenced to display "good" behaviour.

The user management implemented in Shagger makes it possible to observe the behaviour of a user. Making it possible for the network to determine an appreciation for the user's behaviour.

One of the standard services file-sharing communities provide is searching for shared files on the network. On client-server architectures like Kazaa or Napster, searching is done by sending a search query to the central server and receiving the list of files that match the query. Gnutella handles searches completely different as is explained in part I chapter 5.4 "Searching". Shagger's network architecture is, like Gnutella's, completely distributed. Therefore searching will be done in a way similar to searching the Gnutella network. However on the Shagger network different types of searches are possible. First the reason for these different searches is explained and then we indicate how we provide these services.

5.1 GUID-search

A file on the Shagger network is uniquely identified by a GUID. This GUID can be used to refer to a file on the Shagger network. For example: it has to be possible for a web page to offer a link to a file shared on the Shagger network. The link would then look like this:

shag://{3D1BB393-7A08-4c41-A152-E89A808D0A91}

Where shag is identifying the shagger community and the GUID identifies the file the link refers to. This is only one example where we can use references extrnal to the community (in this case on a web page) to refer to a file located on the Shagger community.

A GUID search is not only neccesary to be able to reference files from external applications, but is also needed when a new node connects to the network. As stated in chapter 3 "Data Storage" user data is stored in a file on the network. This file is identified by a GUID. When a user reconnects to the network, the file with his user data is located. The fastest way to do this is to search for the GUID of the file.

For these two uses of GUID searches to be possible a procedure to search the network for a certain GUID must be provided. We showed back in chapter 3 "Data Storage" where data is stored on the network and what different kinds of data are used for. From that chapter we can conclude that we have to search in two different kinds of data: the shared files list and the index list. The shared files list is stored on every node and indicates what files a node shares. An index list is stored at every level one node and indicates where we can find a file shared by one of the node's children. Therefore searching the Shagger network for a GUID is done by searching every node of level one and higher.

The algorithm that implements this search over the decentralized Shagger network makes use of caching. Cashing occurs when a node issues a search query for a GUID and finds a location where the requested file is shared. The Dynamic Naming Service (DNS) [15] provides the same service to map human readable domain names to IP-addresses. When a user wants to navigate to www.yahoo.com his machine contacts the local DNS server. If the local DNS server has no record for this URL it requests that record from its parent DNS server. When one of these servers has a record, it is sent back the way the search request reached the location where the record was stored. On every server the record is kept (cached) and sent back to the child the request originated from. When the same URL is requested again, the local DNS server has the record stored and can directly respond to the query. This way the network

must not be searched again for the same record. In the same way, when a file identified by the GUID is found at a certain location, the record is stored in every index list of all nodes the search request originated from. Figure 5.1.1 depicts such a GUID-search.

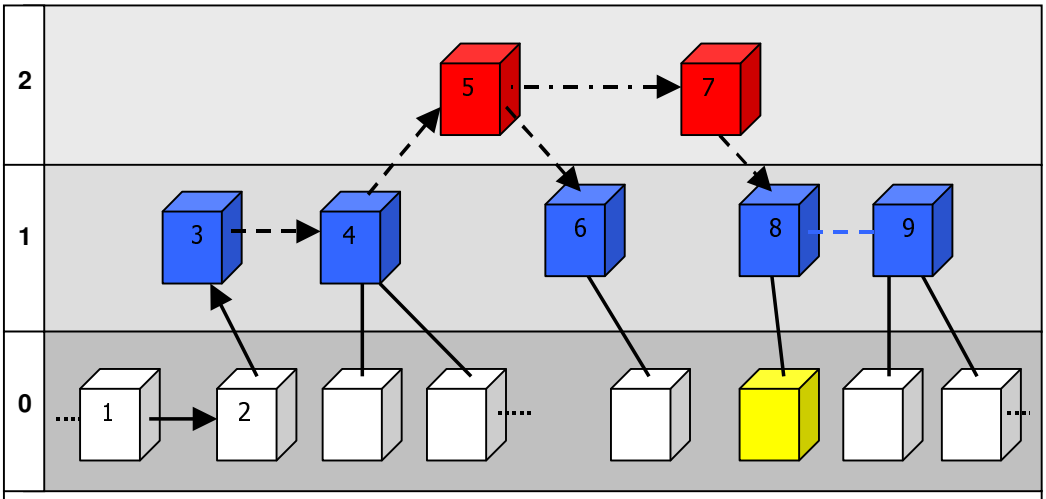


Figure 5.1.1 – *A GUID-search originating from a level zero node.*

Figure 5.1.1 shows the nodes connected to the network. The arrows represent the path a search request originating from the level zero node labeled 1 follows to reach a file shared by the yellow node. We assume there was no caching done by the network prior to this search query. The search query is issued by node 1. The query travels to the logical level 1 parent labeled 3 on the figure through node 2. There the index list is searched and as there is no match for the query, the query is passed on to node 4 of the same level. Again the index list is searched at this node. No match is found and the query is passed on to the level 2 parent of that node. This level 2 node distributes the query to his children and all the other nodes from his level he is connected to. In this example the nodes 6 and 7 are sent the query. Node 6 processes the query, but finds no matches. The search query is not sent on from node 6. Node 7 searches his shared files list and index list, but finds no matches. The search is sent on to node 8. In node 8 the Shared file list is searched and no matches are found. Then the index list is searched and a reference to a node is found. The reference points to the IP-address of the yellow node. The node is contacted and the availability of the file matching the GUID is checked. From that point on the search query is reversed and returns the same path as it came from. This means the result travels through nodes 7,5,4 and 3 respectively. In every one of these nodes the result-record is inserted in the index list. The last stage of the search query is contacting the sending node 1 and supplying it with the IP-address of the node sharing the file searched for.

To illustrate the effectiveness of caching we present the following example: When node 2 performs a search query for the same GUID, his parent level 1 node can respond with the IP-address of the node that shares the file. The node sharing the file is then contacted to check the availability of the file. If the file is still available the extra overhead of searching the remainder of the network has been avoided. When the file is unavailable at the indicated position or the node has disconnected from the network, the search query is again fully executed as in the example prior to caching. However, before resending the search, the correct record in the index lists of every traversed node is invalidated.

5.2 Identity-search

The next kind of search we discuss is used when new files are added to the network. When the files are added to or removed from the shared files of a node, the node's shared files list has to be updated. The shared files list maps GUIDs to files shared by a node. Files are therefore identified by GUIDs. These GUID have to be kept consistent across the network. Two identical files must therefore be identified by one and the same GUID. For this statement to hold the following technique must be used:
When a node connects to the network and it has files shared, every file will have to be checked for its availability on the network. There are two possible scenarios:

1. The file is already located on the network on another node.
2. The file is not found on the network.

For every shared file on the node we must therefore conclude: the file is on the network or the file is not found on the network. If the file is not on the network a new GUID can be assigned to the file. If the file is already available on the network, the GUID for the new file can be copied from the available file. Until now we haven't explained how the availability of a file is determined. We cannot use the GUID search, as the GUID of the file is exactly what we are trying to determine. To determine the availability of a file, we have to search the complete network. This is a very costly operation in terms of network resources. Before we introduce the way we determine the availability of a file we will first discuss how we are going to determine if two files are identical.

Normal searches on file-sharing communities are conducted on file-names or even Metadata stored in shared files. However, this information only leads to fuzzy matches and don't indicate identical files. Two files are identical only if they match byte per byte. This is not per definition the case with two files that have the same filename. To quickly determine if two files are identical we introduce the checksum of a file. A checksum is calculated by using the following properties of a file:

- File length
- Sum of its bytes
- File type

Files are not uniquely identified by their checksum. It is therefore possible for two completely different files to have the same checksum. However the chance of this happening is small enough for our use of the checksum. To be sure two files are identical they will have to be compared byte per byte.

When we want to determine the availability of a file on the network we use the checksum of a file to determine its availability. By searching the network for this value instead of the GUID of a file we can determine if an identical file is available on the network. This kind of search is called an identity search. As we explained in chapter 3 "Data Storage" in every shared file list, not only the GUID of a file is stored. There is also a filename and a checksum. This checksum is used when searching for files identical to the new file.

The identity search we use to determine the availability of a file resembles the GUID-search in the way it operates. When performing a GUID-search on the network caching occurs. Because of this caching not only GUID-identifiers are cached, but also checksums of the files searched for. This will improve performance of identity searches. However, there is one major drawback

associated with this technique: if a file is unavailable on the network a complete network traversal is performed. The checksum has never been cached on the network, as it is unavailable. This could lead to network overloads and introduces a possible vulnerability to DoS attacks.

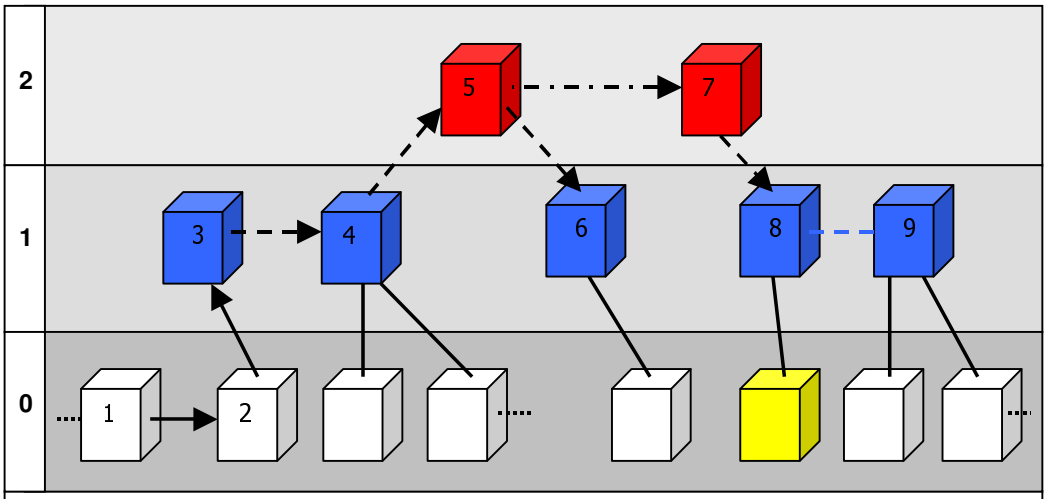


Figure 5.2.1 – *An identity-search originating from a level zero node.*

Figure 5.2.1 displays the path an identity search follows. Note that this picture resembles figure 5.1.1 that displays a GUID-search. In this case the zero level node labelled 1 shares a new file. The availability of the file has to be determined to determine whether to assign a new GUID to the file. The yellow level zero node shares a copy of the file. The network traversal performed by node 1 is the same as is performed in case of a GUID search. Again we assume there has been no caching prior to this identity search. The difference between the GUID-search and the identity search is clearest at the point a file has been found with a matching checksum. The position, GUID, file-name and checksum are then cached in all nodes that form the shortest path to the originating node as happened with the GUID search. However, after returning the result to the requesting node the returned file is further examined to ensure identity of the two files. When identity of the new file and the one located on the network has been ensured, the GUID of the network file is copied to the new file.

To minimize the overload caused by identity searches for files that are not on the network we again use our caching technique. When the complete network has been searched, the node has to be notified that the file is not on the network. In every node the checksum, file-name, new GUID and the associated IP-address of the level 1 node are cached. This way the location of the new node is rapidly introduced onto the network, resulting in better performance of GUID-and identity-searches for that file. However, this will not keep the identity search from having to traverse the complete network.

5.3 Fuzzy-search

All current file-sharing communities offer fuzzy-searches to their users. Users use these searches to locate files of interest on the community. The user has to provide the community application with a string containing information on the file searched for.

As we do not want to limit the kinds of data shared on the community we will not offer different fields to search for. For example: Napster only offers searches on

an author and title field. Therefore the Napster network limits its searches to data that contains an author or a title field in the filename.

On the Shagger network every file has a mapping in a shared files index to the location on the user's hard disk. When a node searches his shared files this list is searched. Therefore only information available in the shared files list is taken into account when searching for files. As we saw earlier the shared files list contains the GUID together with the filename, checksum, Metadata and the type for every file.

When a user wants to find a file he can use two different kinds of searches:

- Simple fuzzy search.
- Advanced fuzzy search.

The simple fuzzy search offers only one field. The content of this field is checked with all the information on a file. For example: a string search is performed on the name and on the Metadata of a file. A file matches a simple search query if one of the strings associated with a file contains one of the search words.

The advanced fuzzy search offers users a more elaborate way of searching for files. The user has to enter a search string that contains tags and search strings. With every tag a user specifies a search term has to be associated. Some standard tags can be provided in the application. For example: the standard MP3 fields: artist, title, year, album, track, etc. However, it has to be possible for a user to extend the standard tags. If a new kind of data is shared on the network with other Metadata tags than the standard ones, these must be searchable.

5.4 Conclusion

In this chapter we introduced the different ways to search the Shagger file-sharing community. Through the GUID-identification of a file we can uniquely identify any file on the Shagger network. This identification scheme has many advantages e.g. in file replication and caching. However, there is an extra overhead introduced to enable the unique identification of files. This overhead is incurred when introducing new files on the network.

6. File Replication

In part one chapter 1.2 "Current Problems and Restrictions" we indicated that file-sharing applications are dealing with two major problems:

- How to improve the availability of shared files on the network.
- How to improve download speed.

Shagger uses file replication to improve the availability and download speed of shared files. File replication is implicitly implemented in all current file-sharing communities. Users that download files share these files and therefore share a replica of the downloaded file. We showed Kazaa was the only community that uses replication to improve the download speed of files. The advantages of implicit replication are:

- Files that are popular are automatically replicated more frequently.
- No explicit replication process has to be provided.

However, there are some drawbacks to this implicit implementation of replication:

- Users can stop sharing a file. For example: the file is downloaded and is taking up to much hard disk space. The user will burn the file onto a CD and delete the copy on his hard disk.
- Users have to download files completely before these files can be shared.

This implicit implementation of replicating files on the network is therefore heavily dependant on the user's behaviour. By making the transfer speed of files and replication strategy dependant on the user's behaviour, it is impossible to control the replication process.

The solution we propose requires extra services to be implemented on the community network and a special treatment of files. That is why we introduced back in chapter 3 "Data Storage" the notion of file chunks. File chunks make a better replication strategy possible.

File chunks are small parts of a shared file. We can now work with parts of a chosen length when replicating. Chunks are made available before a user has downloaded a complete file. Figure 6.1 illustrates this technique. The node labelled 1 on the figure stores the complete file i.e. the 5 chunks that make up the file. Node labelled 2 has already downloaded 3 of these chunks. When node 3 requests the complete file, both node 1 and 2 can start sending chunks to node 3. This approach results in a better performance than first transferring the complete file to node 2 and then transfer the file from node 1 OR 2 to node 3

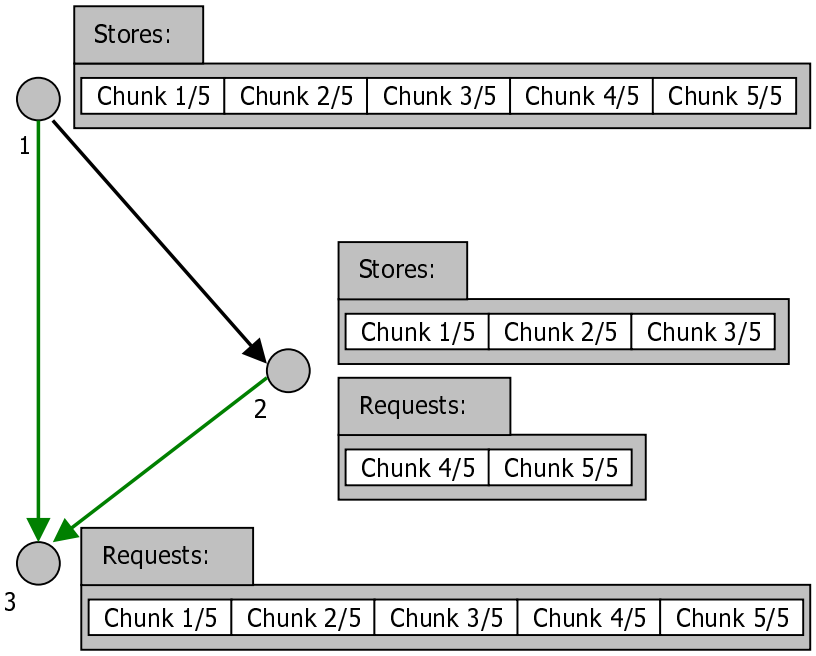


Figure 6.1 – *Example file transfer using chunks.*

For large files this way of downloading files will drastically improve the availability of files. Users that have downloaded only some chunks of a file will be able to supply these chunks to other users. This technique is an improvement over the implicit replication performed by users on current file-sharing communities. By further extending this technique we can make the network responsible for keeping files online and making sure these files are closest to users that request them.

The explicit implementation of the replication strategy in Shagger is made possible though the use of network resources. We have indicated the difference between network and user resources in chapter 2 "Shagger Network Architecture". The network resources will be used to perform replication without a user's interference. The network resources are independent from the user's behaviour with the exception of node disconnection. Users can therefore still determine whether to disconnect their node from the network. However, they can't interfere with what their network resources are used for while their node is connected to the network. Because of this user independence the network can perform replication to improve the efficiency of the network. This is in contrast to current file-sharing communities where file replication is a side effect of the opportunistic behaviour of users.

Before explaining how the network performs file replication, we first determine the different cases where replication is needed.

- When a file is popular, the file can best be available all over the network.
- When a user is interested in a kind of file it is better to locate the files of that kind close to him.

In both cases the community would profit from replication. Replication will reduce the load on specific nodes, reduce download times and improve the search speed.

The community is trying to anticipate the behaviour of users and can therefore offer a better usability than current file-sharing communities.

6.1 Statistical Data

When we want to replicate files so that they are located close to users that are interested in them, we have to know what files are of interest to what user. To do this we will introduce a way to monitor searches and downloads performed by users. When a node issues a search query, the query traverses the network in a depth first fashion. Therefore the query passes through at least one level one node. Figure 6.1.1 displays the two possibilities.

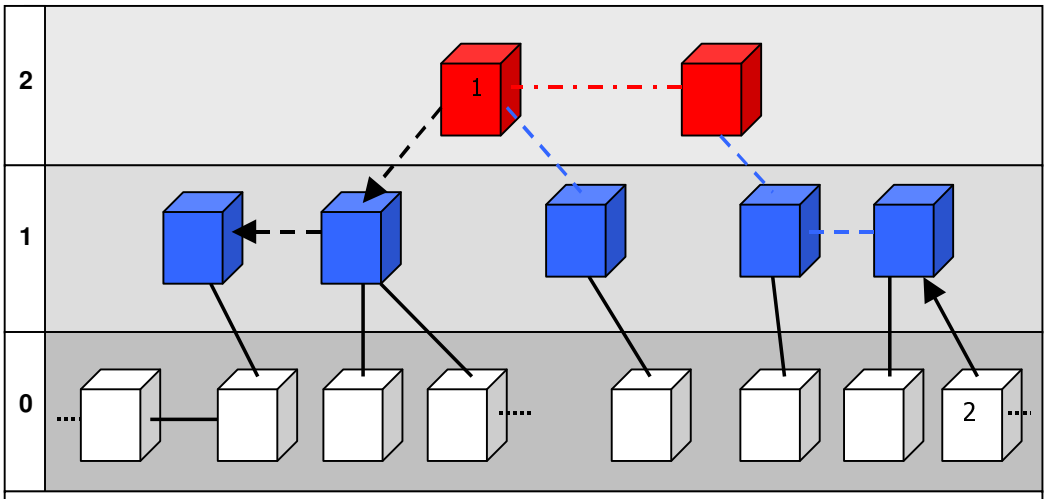


Figure 6.1.1 – *An example network in level representation. A node of a level higher than level one and a zero level node issue a search query.*

Node labelled 1 is a node from a level higher than 1. When this node performs a search query, the search query is sent to one of his children. The query will only return after descending the branch and reaching a level 1 node. The other possibility is that a node of level zero (labelled 2 on figure 6.1.1) issues a search query. In this case the query will also pass through a level one node.

Queries performed by nodes pass through at least one level one node. Therefore the nodes on level one gather data on the searches performed by nodes connected to the network.

Level one nodes are therefore the best location to store statistical data on search queries. We will now proceed with explaining how this data is stored and what this data will be used for.

We have to determine what nodes need to replicate what data. To determine the most popular files, we use statistical information stored in every level one node. When any node on the network is ready to perform replication the following actions are executed:

1. If this node's level is zero: it asks its parent level one node for the statistical information.

2. If this node's level is one: the node has all necessary information present and can determine what file to replicate.
3. If this node's level is higher than one: all level one children of this node are contacted and statistical information is retrieved from all these nodes. This information is then merged.

Every node on the network can determine what file is popular and therefore what file to replicate by looking at the statistical information present at their designated level one node(s).

Replication on level zero is based on statistical information stored in one level one node. However, replication on e.g. a level two node is based on statistical information stored in all its children of level one. Therefore only files that score highest with all of its children are replicated on higher level nodes. Level zero-and one-nodes will replicate files of interest to their neighbor and child level zero nodes. Figure 6.1.3 depicts a sample network in a simplified version of our level notation. As is depicted on figure 6.1.3 the level 2 node labeled 1 determines the file to replicate by evaluating the statistical information in their children that are level one nodes. For the level 2 node labeled 2 this information is stored on just one level one node. Therefore the files this node will replicate are the same that the level one and zero nodes will replicate.

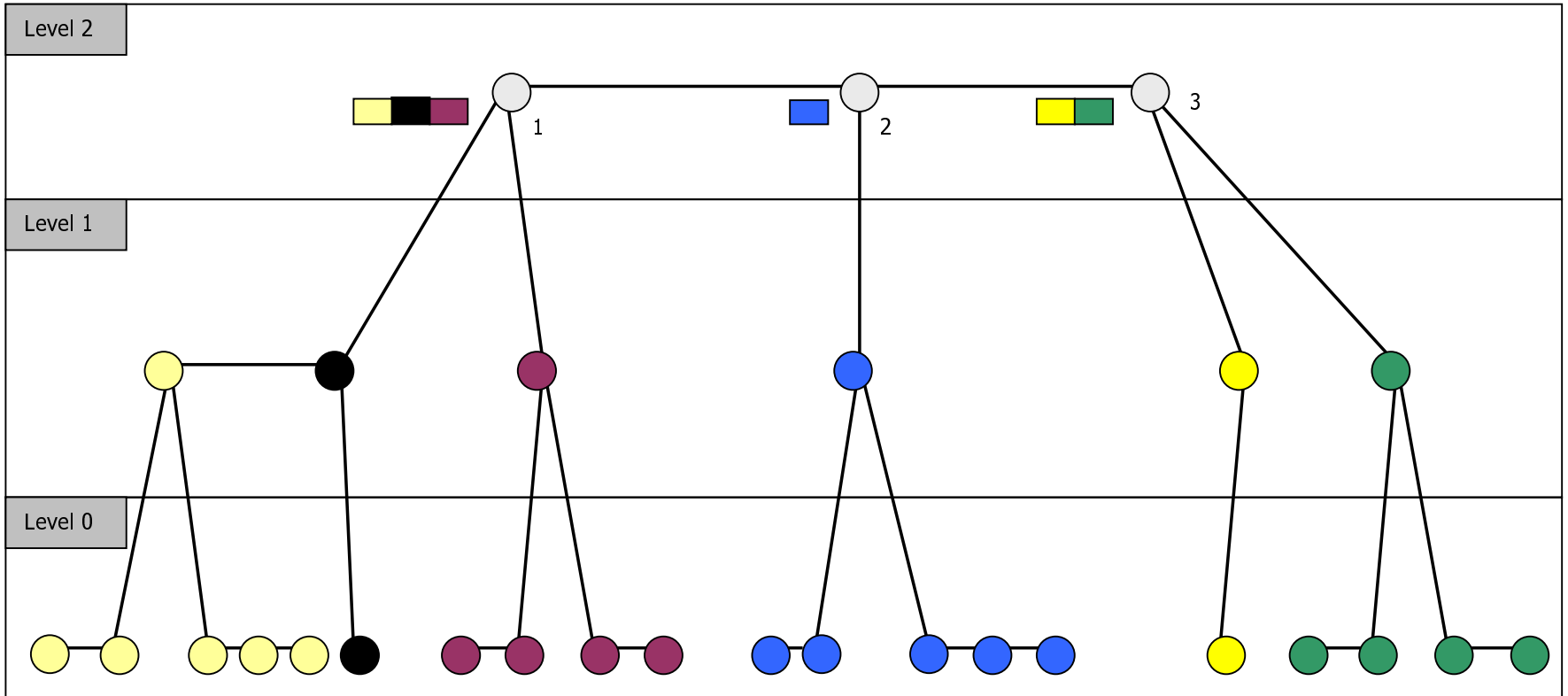


Figure 6.1.3 – *colouring of a simplified level representation matching node usage for replication. Statistics on what to replicate is stored on level 1 nodes for every individual branch. The boxes near the level 2 nodes represent what level one node's statistical data they will use in determining the file to replicate.*

The result of using this method for determining what file to replicate is that files with many transfer requests in every group will also be replicated onto higher-level nodes. Therefore information that is needed by all nodes on the network is replicated on more stable nodes. Information that is only requested by a limited number of nodes is located as near as possible to these nodes by using this technique. However, if there is no demand for a file in a certain number of nodes of a group, this file is not replicated onto that group.

Every node can now determine the level one node(s) it has to contact to determine what file to replicate. We have not yet explained how the statistical data stored in level one nodes is determined. We stated earlier on we would use searches to determine the files users are interested in. This categorisation will have to be more exact then e.g. "the users are interested in audio files". Fuzzy search queries performed by users can lead to a solution: e.g. when a lot of users are searching for audio files of the author "Metallica" the users will probably be interested in a new CD that is published somewhere on the network. The replication technique Shagger uses will first replicate files where they are most popular.

Together with analysing search queries a more direct way for a user to inform the network of his interests can be provided. A user can explicitly submit his interests to the network by using a service more elaborately discussed in chapter 9 "Future Work".

6.2 Flushing

We pointed out that statistical data for a number of nodes is gathered on a level one node. This data is not altered if a node disconnects from the network. Therefore the statistical information could lead to replication of data that was of interest to disconnected nodes.

To solve this problem we delete the statistical data on the level one node after a period of time. The data will then be rebuilt by new search queries performed by nodes and by requesting the statistical data from all nodes that are still connected to the network. We already stated in chapter 3 "Data Storage " that zero level nodes do not keep their own statistical data. Therefore when the statistical data is flushed the files requested by zero level nodes are deleted.

As zero-level nodes tend to disconnect frequently they would influence the statistical data and therefore lead to unwanted replication. Flushing the statistical data on regular intervals solves this problem.

6.3 User Data

A user data-file is typically requested by only one node. Shagger would therefore not place these files in the higher levels of the network, but closest to the requesting node. User data is critical data and we want critical data in the higher-level nodes.

To solve this problem, we use automated incentives to replicate user data. Every node scans its shared files list on a regular basis and user data is identified from the shared files list. GUID searches are performed to access the amount of replicas that are available is high enough for every one of the user data-files found in the shared files list. The higher the level of a node, the more its user data is replicated. The minimum amount of replicas of user data required for a

node is: the node's level times 10. This reduces the chance of losing user data of higher-level nodes.

6.4 Conclusion

We can conclude this chapter by stating that user independent file replication offers improvements relating to transfer speed as well as to availability of files. By offering the user a file availability that tries to adapt to his interests Shagger tries to improve the effectiveness of the community.

The final standard service that is offered by the Shagger file-sharing community is downloading of files. Before a user can download files from the Shagger community he will have to be aware of their availability. A user can check the availability of a file by using a fuzzy-search introduced in chapter 5 "Searching".

7.1 How to Find Results

When a user performs a search he will receive a list of results from the network. This list offers the user an overview of the different matches that were found on the network. The user keeps on receiving results while the network is not fully searched. When the network is fully searched, the user will be notified through the user interface.

The list of results offers extended information about every match that was returned. The extra information was retrieved from the location where the file was found. Extra information can is the Metadata associated with a file e.g. the author of a song, the associated web page or the title of the album the song was published on.

Together with the extra information on a file, the availability of a file is listed. For every source that is found for the file, an extra record will be added. In case only chunks of a file are found at a location, the record indicates what chunks were found. If the complete file is found at a location, the record will indicate that the complete file is available.

7.2 Determine Download Locations

When the user wants to download a file from the results Shagger will try to maximise the download speed. Therefore, the nodes where the file is available, closest to the user, are determined. The Shagger client application will then choose from what nodes to download pieces of the file.

It is clear that nodes that are "closer" to the downloading user in the network architecture will share a larger amount of bandwidth with the downloading user. To determine the locations where the file is available that are closest to the downloading user we propose the following technique:

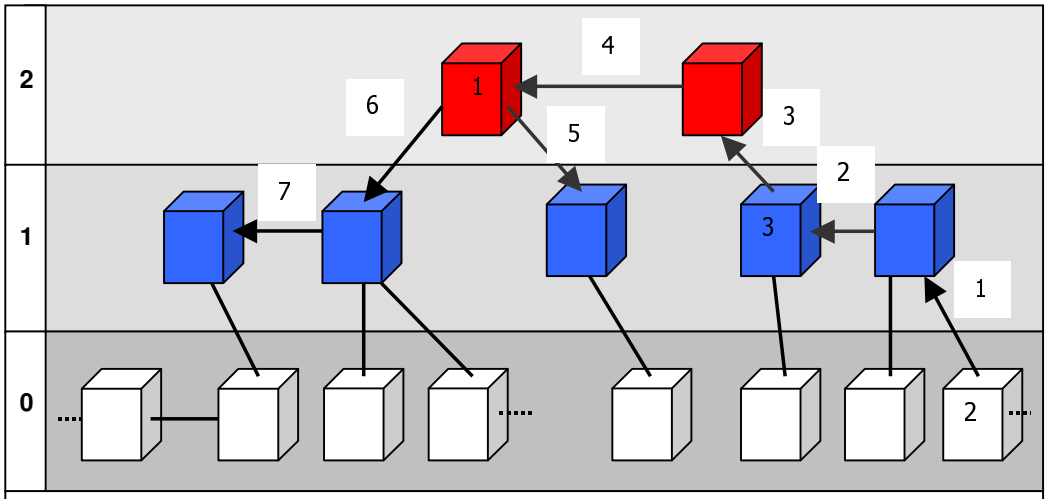


Figure 7.1 – *The order in which a search query traverses the network. The # of hops are displayed alongside the arrows.*

When a node sends a search query, the network will be traversed a depth first fashion. We also know that the Shagger network architecture clusters nodes that share high bandwidths close to each other. We can use these two properties to determine the closest nodes where the file is available for download:

The matches of the file that are closest to him are those that where first returned by the search query he performed. We can not depend on time to define "first". Due to network congestions the message could be delayed. Therefore we have to find another way to determine what result was returned first.
We will use the amount of nodes that have been traversed before the result was found. We can apply this strategy because of the network architecture Shagger uses. Figure 7.1 will clarify this statement.

For example: If node 1 has the file the user of node 2 wants to download, the user's community application can determine whether another node is closer to him. When node 1 sends his results of the query back to the originator of the search query, the number of traversed nodes is sent along. If every node that returns his results to the originating node applies this technique, the requesting node can determine which of these nodes are closest to him. In this case node 3 will inform node 2 that there have been 2 nodes traversed. Node 1 will tell node 2 there have been 4 nodes traversed. This way node 2 can easily determine that node 3 is closest to him.

7.3 Downloading

When the closest nodes sharing the file are determined, the application can start downloading the file from the different locations. The nodes that share the file will be contacted one by one and asked permission to download. This is done until the requesting node's bandwidth is maximized.

8. Error Recovery

In this chapter we will identify the different errors that can occur in the Shagger community and propose solutions for them. This chapter is by no means complete as the possible errors that can occur in a distributed network like Shagger are sometimes hard to spot even when the application is implemented.

8.1 Dirty Node Disconnection.

On the disconnection of a node from the network all information on that node is lost to the network. In chapter 3 "Data Storage" we introduced the different kinds of data that can be present on a node. Here we will discuss what will happen at the dirty disconnection of a node. The consequences such a disconnection has on the network depend on the level of the disconnecting node. We differentiate between a level zero node and a node of a higher level.

8.1.1 Dirty Disconnection of a Level Zero Node

There are two possible cases:

1. The level zero node is directly connected to a level 1 node.
2. The level zero node is connected to another level zero node.

In both cases nodes connected to the disconnecting node are disconnected from the network. These nodes have to be reconnected to the network. They contact the level one node they where indirectly connected to and ask to be reconnected to the network. If their level one node disconnected in the mean time, the nodes will be reconnected in the usual way of reconnecting a new node to the network.

When a level one node receives a message for a level zero node that was disconnected from the network, the message is stored until the node is connected to the network again. If the disconnected node is not reconnected to the network within a certain timeframe, the message is deleted.

There is no data on a level zero node that is critical to the network and therefore no critical information can be lost.

8.1.2 Dirty Disconnection of a Node of Level One or Higher

A node of level one and upward contains more information about the network than a level zero node. Therefore, the procedure for handling a dirty disconnection of these nodes is more complicated.

The different kinds of data that can be present on a higher-level node are:

- File Chunks
- Shared files list
- User data
- Index list
- Service data
- Statistical data

As we are considering the case of a dirty disconnection of a node, we will concentrate on critical data that will be lost.

File chunks are only critical if they are unavailable on the network except on the disconnecting node. Because of the replication strategy applied, if the file is popular, there has to be a replica of this data somewhere on the network. If the data was not available on the network and the file was not replicated, the data is lost to the network. There is no solution for this problem.

A shared files list is built by every node and only contains information on files that node shares. If a node disconnects its shared files list becomes useless and therefore the list is not treated as critical data.

User data is located in a file that is shared by the node. It is therefore treated in the same way as any other file or chunk. If user data is lost there is no way of deducing it from other information and therefore it cannot be regenerated. As user data is the most important data on the network we propose a strategy to overcome loss of user data in chapter 8.2 "Loss of User Data".

An index list is only maintained by a level one node. The mapping the index list represents can be deduced from the level zero nodes that are connected to the level one node. Therefore Index lists can be regenerated.

The node offering a service is responsible for the uptime of the service and for the service data. Therefore service data is not treated as critical data.

Statistical data is located on every node of the network. This data is frequently "flushed" as mentioned in chapter 6 "File Replication". Statistical data is therefore not critical to the network.

Not only data is lost when a higher-level node disconnects. Also the connections the disconnected node shares with other nodes are dropped. This could cause the network to lose its integrity.

A network has lost its integrity if there are two or more halves that are unable to communicate. This is caused by dirty disconnection of nodes from the network. When the network is split up into different parts, these parts have to reconnect. Therefore we propose the following solution:

Every node on the network caches the IP-addresses of the originator of a message. This way when network partitioning occurs, the two halves can try to contact each other. Nodes from which messages have been received are contacted one by one. If the contacted node is unreachable through sending messages across the network, the node belongs to the disconnected part. This procedure is started in both the disconnected parts. We have to identify the exact location where a network split has occurred. Therefore we contact the nodes who's IP-addresses where cached in a top-down fashion. Nodes of highest level are contacted first if these nodes are still connected the next node of a lower level is tried. We can determine the level of a node, as together with the IP-address of an originator of a message, we keep the level of that node.

When the two halves of the network apply this strategy they have a good chance of contacting each other. Figure 7.1.2.1 shows two disconnected halves of a network.

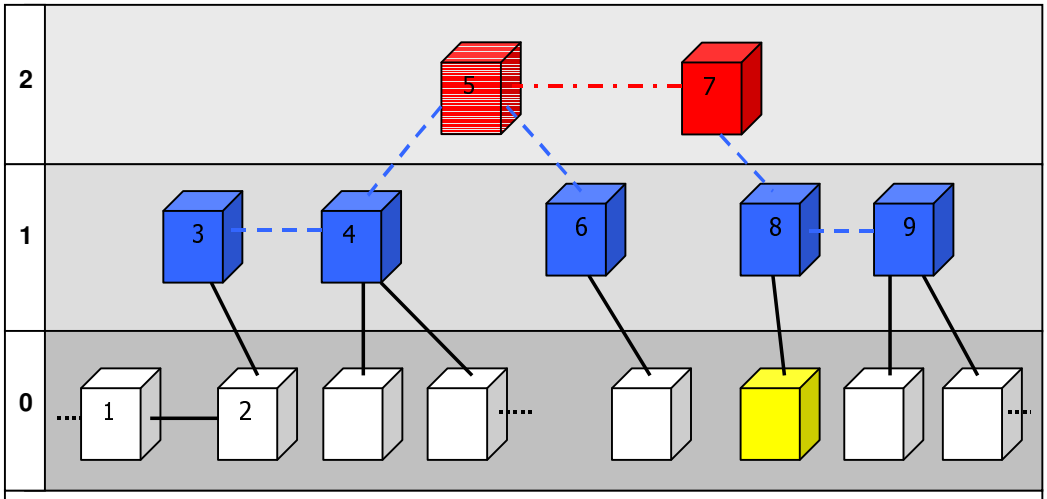


Figure 7.1.2.1 – *Two network halves after node 5 performed a dirty disconnect.*

Node labelled 5 has performed a dirty disconnect from the network. Therefore the network was split up into two parts. It is very likely node 7 has received messages originating from node 3. Therefore node 7 tries to contact node 3 by traversing the network. The node cannot be reached by traversing the network. Therefore node 3 must belong to a disconnected part. The node is contacted through a direct connection and is asked to reconnect to the network. Note that in the case of network partitioning the part with the node with the highest level is regarded as the network. Other parts with nodes of a lower level are regarded as a disconnected part. When a part of a network is disconnected and is asked to reconnect, all nodes are one by one reconnected to the network starting with the nodes of the highest level.

The problem has been reduced to providing the network with a way to identify a network split. When a node is disconnected that liked other nodes to the network we can assume a network split. The amount of nodes that are disconnected from the network in this way varies with the level of the disconnecting node. Nodes that can no longer reach higher level nodes assume there has been a network split and start traversing the network.

8.2 Loss of User Data

When user data is lost, there is no way of rebuilding it. The user that loses his user data has to be reconnected to the network at level zero. To avoid this setback the Shagger community application can keep track of the previous GUIDs identifying his user data. This way, if the current user data is missing from the network, the user can use older user data to reconnect to the network. Therefore the user will not lose all of his advantages when reconnecting. This strategy can be applied with the 3 former user data GUIDs. As mentioned in chapter 4 "User Management" the user data older than 3 versions is deleted from the network.

We pointed out some features Shagger offers to its users that are not available in current file-sharing communities. In this chapter these former mentioned services will be further elaborated and new services will be introduced to show the improvements possible to the standard services offered by the Shagger community. Users of the file-sharing community cannot directly use the replication or the user management service. However extended services can make use of these standard services. All of these services are only provided to illustrate the possible future extensions the Shagger file-sharing community can offer.

9.1 External Referencing

External referencing is used to interlink standard web pages with the Shagger file-sharing community. This service allows users to post files located on the community on standard web pages on the Internet. A user that wants to download files from the Shagger network will have to install the Shagger client application. Through interlinking the user's web browser with the Shagger client software links between the WWW and the Shagger community are possible. A normal link on a website looks like this:

http://www.yahoo.com/main.html

When looking at this URL we can distinguish three different parts:

1. http://: the protocol used to connect to this resource.
2. www.yahoo.com: the name of the server where this resource is located.
3. main.html: the name of the resource to retrieve form the server.

One of these values is not applicable to Shagger. The address of the server is not needed because the Shagger client installed on the users PC can search the entire community. There is no way of specifying a specific node on the Shagger network. A Shagger Resource Locator looks like this:

Shagg://{45AE05CB-73B9-401c-B6B5-81980EAF1FCD}

Shagg is the protocol Shagger uses. Because of this value the browser will contact the Shagger client application. The value {45AE05CB-73B9-401c-B6B5-81980EAF1FCD} is the GUID uniquely identifying a resource on the Shagger network. The browser passes this GUID on to the Shagger client application. The Shagger network is then searched through a GUID-search and the requested file is queued for download.

9.2 Self Served Pages

Every node from level one upward can serve web pages. These pages are not published on the WWW, but on the Shagger community. Current file-sharing communities offer no way to let users profile themselves on the community. Shagger offers Self Served Pages (SSP) for this purpose. Users can publish information on themselves, on the files they are looking for or on files they are offering.

SSP can be used to enable other users to make file requests or even start a little cooperation with a message board and such. The same possibilities can be offered as on normal web pages. The information on these SSP is not replicated on the network. The user must see to the uptime of his site by keeping is node online. References between SSP on the Shagger network can be made though a special kind of URL. For example:

SSP://134.184.118.132/main.ssp

Would point to the SSP called main.ssp on the PC with IP-address 134.184.118.132. These pages can contain a server side scripting language and a limited client side scripting language offering users the flexibility of dynamic pages.

9.3 File Requests

File requests to other users can be made on your SSPs. However a user of the Shagger community can also make these requests toward the community network.

Because of the replication technique the network uses, a user can explicitly specify what file he is looking for. For example: a user would really like to download a complete Linux distribution with a total size of 600MB. Alas this file is not completely available on the network at this time. There are 5 chunks missing from the file and no one in the Shagger community has these chunks shared. Instead of demanding from a user to keep his PC online, he can issue a File request, for the chunks, to one of his level one nodes. From the moment a user has issued such a request and a node in his proximity wants to replicate a file the level one node will propose one of the chunks requested by the user. This way when the user reconnects to the network, there is a possibility that the right chunks were available long enough to be replicated close to him. He can then download these chunks and complete his file.

File requests can also be made prior to downloading a file. If the network is given time to replicate the requested files, bringing them closer to the requester, the download speed will be drastically improved.

Due to the new techniques we used to improve the overall effectiveness of the Shagger file-sharing community new problems are introduces.

10.1 Initial Connection to the Network

The Shagger network suffers from the same problem the Gnutella network suffers from: there is no initial point of contact for a new node to connect to the network.

The Shagger network is a distributed network. Therefore there is no central server that can be used as initial connection point for new nodes. The users of the Gnutella network have solved this problem by setting up web pages where IP-addresses of nodes connected to the Gnutella network are posted. This information has to be updated frequently as the Gnutella network is in constant change.

The same technique could be used with Shagger although it is not user friendly to let a user search for an IP-address somewhere on a web page.

An alternative would be letting nodes on the Shagger network serve a web page. If this web page would be indexed users could use the WWW to search for a Shagger node. The Shagger application could also be offered for download on these pages. However, this would introduce a vulnerability to the network. Every node that serves a page can be found through a simple web-search. Therefore they form an easy target for hackers or for DoS-attacks.

10.2 Overloading the Network

Sometimes a high number of nodes is disconnected from the Internet. For example: through a failure of their Internet provider. If these nodes would all reconnect at the same time, the Shagger network would be overloaded with reconnection requests. A possible solution would be to queue reconnections on top nodes. A negative side effect of this is that it would lead to long connection times.

10.3 Community Overview

The Shagger network fails to offer users a clear view on what users are located close to him. By implementing extra services this problem could be solved but there is no standard support for this feature in Shagger.

A good extra service that could solve the problem would be a forum where only users of a certain level and who share the same top-level node can read and post messages. This kind of forum could be implemented by using the SSPs introduced in chapter 9 "Future Work".

10.4 New Files Online

When any node connects to the network his shared files are checked for availability on the network. When a new file has to be checked for availability, a complete network traversal has to be performed. Therefore, connecting users with a high amount of shared files to the network could cause a serious overhead on search requests.

However, by means of caching and the fact that the amount of new files on the network will drop, the overhead of the identity searches is reduced.

This chapter will recapitulate what we proposed in this thesis and we will evaluate the Shagger file-sharing community against the current file-sharing communities we introduced in part one. Future improvements and extensions will also be proposed in this chapter.

11.1 Overview

The primary goal we stated when starting this thesis was: improve the quality of service of file-sharing communities. In this thesis we introduced techniques to solve problems current file-sharing communities are dealing with. This chapter offers a high-level overview of the techniques we introduced. It will also show how the different services interlink to provide a better overall service.

Network Resources:

In chapter 2 "Shagger network architecture" we introduced network and user resources. By introducing network resources, we enable the network to act on it's own account. Therefore, actions performed by the network are beneficial for the complete network and not only for some of the users. All of the users grant the network usage of a small part of their resources so the network can improve the overall effectiveness of the community.

Levels:

We needed a way to determine the right location for a piece of data. By classifying nodes into levels we could use the properties of a piece of data to determine where this data would best be stored on the network.

User Data:

By classifying nodes into levels we are classifying users in behaviouristic classes. To enable us to classify users according to their behaviour, we have to study users. Therefore a way is needed to keep information on the behaviour of a user. We solved this problem by introducing user data. User data is Information about a user and it is stored in a file on the network.

User Management:

Through user data and levels we can study the behaviour of users on the Shagger network. We now need a way to influence the behaviour of a user. This is done by associating the levels a user can reach with rewards and therefore at the same time punishments. This way the network influences the behaviour of users.

GUIDs:

We introduced GUIDs to identify files in chapter 5 "Searching". Using this GUID exact searches are possible. GUIDs also introduced the possibility for a user to communicate with the network about a given file. Users can now tell the network something about a file. For example: I downloaded

that file or I would really like to download that file, but the transfer speed is to low. Through this communication the network can be made aware of preferences or needs of users.

Statistical data:

Through GUIDs a user has a language to "speak" to the network. Through statistical data we gave the networks ears to "listen" to users. The network tries to adapt itself to the users through the automatic gathering of statistical data. We also sketched a direct way for users to communication to the network through File request introduced in chapter 9.3 "File Request". Through user-network communication the network can provide a better quality of service to the users.

Replication:

The network may know what users like and don't like, but we have to give the network a way to act on the user's preferences. Therefore replication was introduced. Users explicitly perform replication when they download files. However, Shagger also performs implicit replication. This kind of replication is only performed by the network. Statistical data holds "commands" communicated to the network by users. The Implicit replication performed by the network is based on statistical data. Therefore, by performing implicit replication, the network executes the "commands" of the users. To perform the implicit replication the network uses network resources.

11.2 Evaluation

In this chapter we evaluate Shagger by use of the same table as in part one, chapter 6 "Overview", to compare Shagger to Napster, Kazaa and Gnutella.

Community	**Napster**	**Kazaa**	**Gnutella**	**Shagger**
Property				
Network architecture	Client-server	Client-server	Distributed	Distributed
Search speed	Almost constant	Almost constant	Dependant on network extent	Dependant on network extent and caching
Complete list of results	Yes	Yes, but limited to 200 results per search	Dependant on network extent	Yes
Replication technique	Implicit	Implicit	Implicit	Explicitly implemented
User management	Centralised	Centralised	Absent	Distributed and behaviouristic
Fairness	Everyone equal	Everyone equal	Unfair opportunities for users (searches)	"Good" users get better service from the network
Data storage	Centralised	Centralised	Decentralised	Decentralised
Scalability	Through external intervention	Through external intervention	Automatically (However, badly implemented)	Automatically
Implication of node failure	In case of server: network down	In case of server: network down	Network can still function normally	Network can still function normally
File downloads	From one source	From multiple sources	From one source	From multiple sources
Compatibility with file-types	Mp3 only	All types	All types	All types
Level of detail of search results	File size, bit rate, name	Title, author, file size, different sources, etc	File size and name	Dependant on Metadata stored in files
Download speed estimation	No	Yes	No	No
Network balancing	Not applicable	Not applicable	No balancing	Yes through levelling
Adaptation to users	No	No	No	Yes
Initial Connection to the	Through central	Through central server	Through any connected node	Through any connected node

network	server			

11.3 Conclusion

From the overview and evaluation we offered in chapter 11.1 and 11.2 it is clear that Shagger offers more and better services than current file-sharing communities. Throughout the different chapters we used properties of file-sharing communities to offer a better quality of service. To make use of these properties we had to implement new services and provide extra data. For example: Statistical data and GUID-identification to perform adaptive replication. All of these services where made possible by introducing network resources, making the services independent of user behaviour. We discussed the problems caused by the new services and the overhead they introduced. In chapter 9 "Future Work", we illustrated how extra services can be implemented over the standard services.

We can conclude this thesis by stating that the goals we put upfront are certainly met by the Shagger file-sharing network.

Bibliography

[1] **What are the characteristics of client/server architecture?**
http://www.faqs.org/faqs/client-server-faq/section-14.html

[2] **Yahoo!: Technical Case - A Tale of Two Thousand Servers**
http://www.intel.com/eBusiness/casestudies/yahoo/techcase.htm

[3] **Widening The Broadband Bottleneck**
http://www.zdnet.com/intweek/stories/news/0,4164,2559117,00.html

[4] **Clip2 : Gnutella Hits the Wall**
http://dss.clip2.com/gnutella.html#q3

[5] **Napster**
http://www.napster.com

[6] **O'Reilly Open P2P**
http://www.openp2p.com/

[7] **Seti@Home**
http://setiathome.ssl.berkeley.edu/

[8] **Kazaa**
http://www.kazaa.com

[9] **Kazaa - Our technology**
http://www.kazaa.com/index.php?page=technology

[10] **Gnutella News**
http://www.gnutellanews.com/information/what_is_gnutella.shtml

[11] **Gnutella: Alive, Well, and Changing Fast**
http://www.openp2p.com/pub/a/p2p/2001/01/25/truelove0101.html

[12] **Gnutella: To the Bandwidth Barrier and Beyond**
http://dss.clip2.com/gnutella.html

[13] **The GUID Number**
http://www.searchlores.org/fiatlu/GUIDnumber.html

[14] **Andrew S.Tanenbaum: Distributed operating systems**
Prentice hall 1995,p416

[15] **DNS overview and general references**
http://www.dns.net/dnsrd/docs/whatis.html