MSc Distributed Computing Systems Engineering

Department of Electronic & Computer Engineering

Brunel University

Distributed Software Installation for Free Software

Adrian Reber

Dr. T. J. Owens

08/2004

A Dissertation submitted in partial fulfilment of the requirements for the degree of Master of Science

MSc Distributed Computing Systems Engineering

Department of Electronic & Computer Engineering

Brunel University

Distributed Software Installation for Free Software

Student's name:_____

Signature of student:_____

Declaration: I have read and I understand the Department's guidelines on plagiarism and cheating, and I certify that this submission fully complies with these guidelines.

Acknowledgements

I would like to thank Dr. T.J. Owens for accepting and supporting my dissertation topic and Prudence Lawday for her excellent proof-reading. I would also like to thank Alexander König for his docbook templates and answers to my docbook related questions and non-docbook related tips and ideas. Thanks to Warren Togami for applying the changes to download.fedora.us required for this dissertation and to Axel Thimm for his feedback on parts of this dissertation. Of course, I am thanking my parents Vlasta and Wolfgang for their perpetual nagging how far my dissertation is and thus pushing me to finish it.

The biggest gratitude goes to my wife Jennifer who always believed that I will finish this dissertation and supported me always in every possible way. Jennifer, this is for you!

Table of Contents

1. Introduction 1

1.1. Introduction 1

1.2. Aims 2

1.3. Objectives 2

1.4. Methodology and Limitations 3

1.5. Dissertations Structure 4

2. Background 5

2.1. Background 5

2.2. Initial Survey 7

3. Design and Methods 10

3.1. Initial Design 10

3.1.1. Database 10

3.1.2. Monitoring 11

3.1.3. Client Side 13

3.2. Database 13

3.3. Mirror Assessment 16

3.3.1. up-to-dateness 17

3.3.2. Best Mirror 21

3.4. Security 23

3.4.1. Password Handling 23

3.4.2. HTTP Referrer 24

3.4.3. SQL Injection 26

4. Results 28

4.1. Programmes and Programming Languages Used 28

4.1.1. Programmes 28

4.1.2. Programming Languages 29

4.2. Tools written 30
4.2.1. checkmirrors.pl 30
4.2.2. mirror-setup.py 37
4.2.3. Web Interface 39
4.2.4. checkmirrorpermission.pl 41
5. Conclusions 44
5.1. Overall Aims 44
5.2. Implementation Specific Goals 45
5.3. Limitations 46
5.4. Final Conclusion 47
Glossary 49
References 54
Bibliography 56
A. Management of the Project 58
A.1. Time Management 58
A.2. Project Management 60
A.3. See It For Yourself 61
B. Source Code 62
B.1. checkmirrors.pl 62
B.2. mirror-setup.py 69
B.3. Web Interface (functions.inc) 76
B.4. checkmirrorpermission.pl 79

List of Figures

3-1. Monitoring 11

3-2. Database Design 14

3-3. Mirror Setup GUI 19

3-4. Mirror Status Page 20

3-5. Protocol Selection 25

4-1. `checkmirrors.pl` flowchart 31

4-2. mirror-setup.py Classes 37

4-3. My Mirrors 40

4-4. `checkmirrorpermission.pl` flowchart 42

A-1. Gantt Chart during Interim Report 58

A-2. Final Gantt Chart 58

Chapter 1. Introduction

That's the problem with science. You've got a bunch of empiricists trying to describe things of unimaginable wonder.

Calvin

1.1. Introduction

Most currently available Linux Distributions which have a large user-base are backed by a commercially operating company. The one big exception to this pattern is Debian [1], which is run entirely by the Linux community. There are of course also many other Linux Distributions not backed by any company, but Debian is the one with the largest user-base. The great advantage of such a community-based model is that it is very easy for new software to be added to this Linux Distribution as many people are involved and it is not hard to find a maintainer for the new software package.

Red Hat [2] is an example of a commercially based Linux Distribution. Red Hat is one of the few Linux Distributions, if not the only one, which works independently and makes a profit using *Free Software*.

For a distribution like Red Hat, it is not particularly easy to include software in its distribution as it lives from supporting the Red Hat Linux Distribution and a new software package means an additional part which needs to be supported and therefore requires additional manpower involving additional expenses.

As the community-based approach works very well and efficiently for Debian,

some people started something similar, but based on the Red Hat Linux Distribution. As all the parts of both Linux Distributions are based on *Free Software*, the usual approach for the software installation is to download it from any of the available *mirror servers*. As Debian is already 10 years old, the infrastructure for software distribution and automatic installation already exists. For the community-based parts of Red Hat, however, such a complete infrastructure does not yet exist, and this is the starting point of this dissertation. Of course, there are already some ideas on how to handle the software distribution. At present, there is still a lot of development to be done, so this is a good time to offer a complete solution which is to be developed in this dissertation. Most of the current mechanisms are based upon ideas from Debian or from experience with processes which did not work as expected and therefore do not need to be tried again or, if so, only in an improved form.

1.2. Aims

The following is a list of the aims which motivated me to research this topic and which I try to achieve:

- to develop or create a system using only *Free Software* and the help of the *Free Software* community if necessary;
- to develop or create a system supporting the *Free Software* community;
- to use existing expertise as well as to learn new techniques to reach the above aims.

1.3. Objectives

During the initial survey in Chapter 2, the following objectives to fulfil the aims stated in Section 1.2 were identified by many discussions and material found in literature:

- **Database** - A database will be the central part of the whole system. The database will, of course, not be developed but used as a storage for the necessary data. The focus of this part will be to offer an interface to the database for the users and administrators.
- **Monitoring** - Based upon the entries in the database, another part of the system will monitor the *mirror servers*' availability and data integrity. This will be a periodic process which will be run regularly, for example every hour.
- **Client Side** - The parts which are developed for the client side are still dependent on how the new Fedora Linux Distribution develops. On all accounts and independent of the distributions' development, a tool will be created which lets the user select which *mirror server* should be used for the software installation. This tool needs to offer a command line interface as well as a graphical user interface.

1.4. Methodology and Limitations

To achieve the aims in Section 1.2, the following methodology is used. A programme solving the task will be implemented and analysed. This is already where the limitations are. Each programme implemented during this dissertation will always be in a prototype state. This is due to the fact that time for the implementation is always limited and although it is the goal of the dissertation the deployment of the programmes created will always be very limited. This will be at least the case during the dissertation. A wider deployment of the results will probably only be

available after the dissertation has been finished.

1.5. Dissertations Structure

This dissertation is structured as follows:

- Chapter 1 contains this introduction.
- Chapter 2 provides the background on the basis of this dissertation and tries to formulate the aims and objectives.
- Chapter 3 describes the design and methods on which the results of this dissertation are based.
- Chapter 4 supplies information on the implemented software, on why and how something was programmed. This chapter explains the details of the design and structure of programming results without going into every detail of the programmed code.
- Chapter 5 provides the conclusions drawn by this dissertation.
- *Glossary* contains the glossary.
- *References* contains a list of referenced information.
- *Bibliography* contains the bibliography.
- Appendix A contains a short description of the time plan of this dissertation and the *Free Software* used to create all the parts of this dissertation.

Chapter 2. Background

I like to say “quark”! Quark, quark, quark, quark!

Calvin

2.1. Background

One of the big problems in software distribution, and this is not only the case for *Free Software*, is the dependence of one software package on another. The most common case is that one program requires multiple shared libraries which are linked dynamically during runtime. Usually, the package manager will check the local database containing the installed software to find out whether all the software required for the current package is already provided by the system; if this check is positive, the software will be installed. If the check is negative, the software will not be installed and a message will inform the user which requirements for the software installation have not been fulfilled by the system.

The solution to this problem is that the package manager on the local system has to have a database of all packages available on the *mirror servers*. In this way, the package manager can resolve all the dependencies by downloading all the necessary software packages and installing them in the best order. As this is not really the right task for the package manager, the most logical solution is to write a front-end for the package manager which uses the package manager to install new software but has its own database which it uses to resolve all the dependencies. This is exactly what has been done with Debian.

Debian uses *dpkg* as its package manager and *apt* is the front-end which handles all the dependency resolutions, ordering the package and, if necessary, downloading

the package from any of the Debian *mirror servers*. The tool *apt* works very well and is one of the reasons Debian is so popular. It is therefore not very surprising that the users of Red Hat demanded something similar. As *apt* is *Free Software* and the source code is of course available, it is obvious that *apt* was changed so as to work on Red Hat-based systems. This is exactly what Conectiva [3] did. Conectiva is also a Linux Distribution which is based on the package manager from Red Hat: *rpm*.

It was not long after Conectiva released their port of *apt* for *rpm*-based Linux Distributions that people started to use *apt* not only with Conectiva Linux but with many other *rpm*-based Linux Distributions. The first step in using *apt* with *rpm* was to make some of the *mirror servers* for Red Hat *apt*-aware. This enabled everybody who wanted to test *apt* to install any package available in the distribution. Now that every package available in the distribution was only one command away the demand for bigger software repositories, just like Debian, started to arise. As nobody expected Red Hat to create such software repositories, some people started creating software repositories on their own. The following is a list of the larger and better-known repositories:

- http://freshrpms.net/
- http://www.atrpms.net/
- http://newrpms.sunsite.dk/
- http://apt.sw.be/

Repositories like these were the first step in providing a larger number of software packages. Although these repositories offer a good selection of high-quality packages, there were still some problems. In the beginning, all these repositories were developed independently and a lot packages were maintained in multiple repositories. The repositories became incompatible as a result of different naming schemes

of the packages and similar packaging-related problems. Consequently, a user either had to change the configuration which repositories to use often or just opt for one of them and not use the others. Another problem was that each of these repositories was maintained by another person who had personal preferences and usually only packaged what they needed. Of course, the maintainers responded to package requests but some people were unhappy that there was not any really well defined process on how to add a package to a repository. This is also one of the big advantages of Debian. Each package is usually only maintained by the person who is interested in it and therefore most of the packages are very well maintained.

That was the point at which a group of people decided that, if it was possible for Debian to maintain a extremely large pool of packages with well defined processes for package creation, bug reports and quality assurance, this must of course also be feasible for a software repository based on Red Hat. This was the birth of the Fedora Project [4].

The goal of the Fedora Project is to create high-quality third-party add-on packages for the Red Hat Linux Distribution. To achieve this goal, many discussions were necessary to decide on many of details. Maybe there were too many discussions, but the goal was to define rules and processes for every detail of the Fedora Project, so that it would not be necessary to learn from mistakes. After some months, the Fedora Project was finally ready to start distributing software packages.

The rules for package creation, package maintenance and quality assurance were clear and Fedora started producing *rpm*-packages; it was not long before some *mirror servers* were found which offered help in distributing Fedora. As everything in Fedora was well planned, there was of course the desire to structure the mirror system as well as possible. This is the point where this dissertation starts: discussing the development of a *Distributed Software Installation for Free Software*.

2.2. Initial Survey

As the goal is to develop something to support a community-based *Free Software* project, this dissertation will of course be developed as *Free Software* and also be based on *Free Software* as much as possible.

A very surprising and rather unanticipated change in Red Hat's product policy happened which led to the merger of the Fedora Project and Red Hat. Red Hat will no longer sell a cheap consumer-oriented Linux Distribution but will only offer an enterprise oriented Linux Distribution with the name Red Hat; parallel to this, it will develop a community-based Linux Distribution with the name Fedora [5]. Although this merger happened, the former Fedora Project will still continue to exist, but it will act as a filter through which new packages will enter the new Linux Distribution named Fedora.

The first and main part of the whole system developed during this dissertation needs to be a database, in which every mirror maintainer enters the information about his *mirror servers* and himself. The main reason for having information about the mirror maintainers as well is that, in some rare cases, it may be necessary to reach them if, for example, a very important security update is necessary or the *mirror servers* have not been reachable over a longer period. This database will then be the basis of the other parts of the system.

An important part is the monitoring of the *mirror servers* to ensure that the list of available *mirror servers* is accurate and the user does not have to start searching for the best server. The monitoring should not only include the test that the *mirror servers* is available and up-to-date but it should also verify the integrity of the repository. A possible way of testing the integrity is to use *GnuPG* to sign parts of the repository with a private key and the integrity can then be verified by the public key.

The monitoring of the *mirror servers* should lead to a list of servers which are reachable and up-to-date. This list is then the basis upon which each user can update his system or install new software. The user can select his preferred *mirror server* from this list. In addition to a manual selection of the best *mirror server* by the user, an automatic mode might select the best *mirror server* by measuring which server is the best for the current user. This can be achieved by measuring which of the available servers offers the best bandwidth usage or has the best reachability in terms of network distance.

Another part might be how to handle the initial installation of a front-end to the package manager which handles the dependency resolution. This largely depends on whether the new Linux Distribution Fedora will include a tool like *apt* or something similar.

Chapter 3. Design and Methods

If you couldn't find any weirdness,

maybe we'll just have to make some!

Calvin

3.1. Initial Design

After the definition of the aims and objectives, the following initial design has been created:

3.1.1. Database

This part requires two steps. The first step is to determine a database design including all the necessary tables, the second step consists of developing of the user interface for the database. The design of the user interface depends on the database design and the database design depends on the user interface. As one aim was to offer a user interface which can be accessed from all over the world, the decision to offer a web-based interface was inevitable. This decision also influenced the database design as, with a web-based interface, it becomes necessary to offer user management. A user management independent of the one from the system offering the web interface has the advantage that these users do not require full access to the system. This (giving users full access to the system) could, of course, be a security problem and therefore the user management is completely independent of the system it is running on. There is, however, also a disadvantage in using this design. It requires the user authentication part to be developed again although something like this has been developed many times before. As the data stored in this database are not really

valuable, the authentication uses a simple design.

In order to secure not only the transmission of the user name and password but also the transmission of all the data against eavesdropping, the web-based part of the application should only be used with *HTTPS*. This offers a basic security using the resources provided by web servers and browsers. To provide additional security for the passwords, these are not stored as clear text in the database but as a *MD5* hash.

The user web interface will be written in *PHP*. *PHP* "is a general-purpose scripting language" [6], but is also very well suited to embedding in HTML. This is one reason for using *PHP*, but the database support also makes *PHP* a very good choice. It is also the goal to develop a command line front-end for some parts of the management of the data about the mirrors. The reason for a command-line interface is that it is much easier to add many new users to the system on the command-line than through a web interface, as a command-line tool can be used much better in scripting. It has not yet been decided whether this command-line interface to the database will also be written in PHP as this depends on the programming language used for the other parts of the whole system. Possible languages are *Perl*, *Python* and *PHP*.

3.1.2. Monitoring

To monitor the available *mirror servers*, a connection will be established to each of these servers. During this connection, the main server will try to read a certain file from the *mirror server*. This file will contain a time-stamp from which the main server can determine when the *mirror server* has synced its data with the main server.

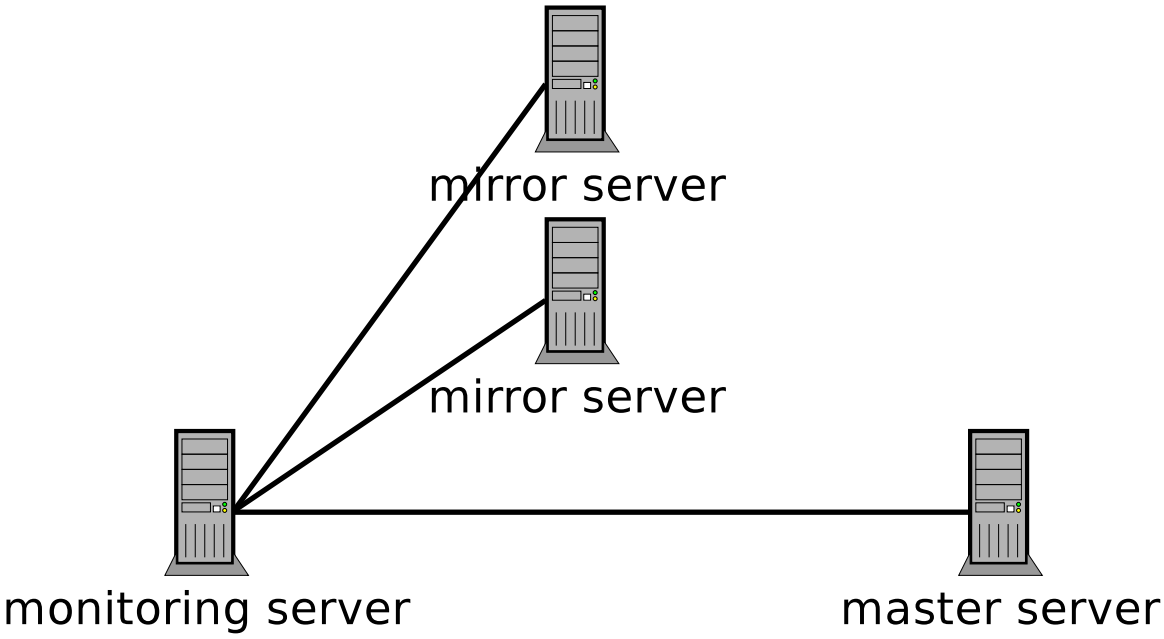


Figure 3-1. Monitoring

The file containing the time-stamp will be generated at the main server at a certain interval (every 10 minutes, for example). A possible extension to the time-stamp file is to sign the file digitally as a check on the integrity of the *mirror servers*. The problem with this approach is that the integrity check is only carried out for the file containing the time-stamp and therefore does not guarantee that the remaining data of the *mirror server* are still valid.

The check of each *mirror server* will probably be made every hour. This depends on the number of *mirror servers* available as it is important that the checking of the *mirror servers* should not require too many resources, and one hour seems a reasonable interval for checking quite a large number of *mirror servers* It is also important that the timeout after a *mirror server* is marked as unreachable is rather small so that not all the time necessary for checking the availability is spent waiting for servers which are down. The feature of monitoring the *mirror servers* every

hour will not be part of the monitoring process but will be implemented by using the already existing cron daemon.

Another idea for improving the performance of the monitoring is to parallelise the monitoring process. It is, however, important that the level of parallelisation does not exceed a certain amount of processes so that the monitoring server is still able to work normally and does not break down under the high workload.

The monitoring process will be developed using either Perl or Python as the programming language, but this still depends on the other components. At present, it is more likely that it will be developed using Python.

3.1.3. Client Side

The client side will consist of a tool connecting to the main or to the monitoring server and requesting the list of available servers. The user then has to select one of the available mirrors from which the software should be downloaded from this point on. The user can also use this tool to check which of the *mirror servers* is closest with the best connection for the user. The closest mirror will be the one with fewest network hops and the shortest time between *icmp*-echo-request and *icmp*-echo-reply.

At present, the goal is to offer a command-line version as well as a *GUI* version of the client-side tool. The *GUI* version, however, will be developed first. During the design and development, this will be an important point so that the logic of the tool will be separated from the view and it will thus be easy to replace the *GUI*-based view with a command-line based view if required.

3.2. Database

The initial database design described in the Interim Report was updated due to requirements emerging during the development of this dissertation. The final database design includes the following tables:

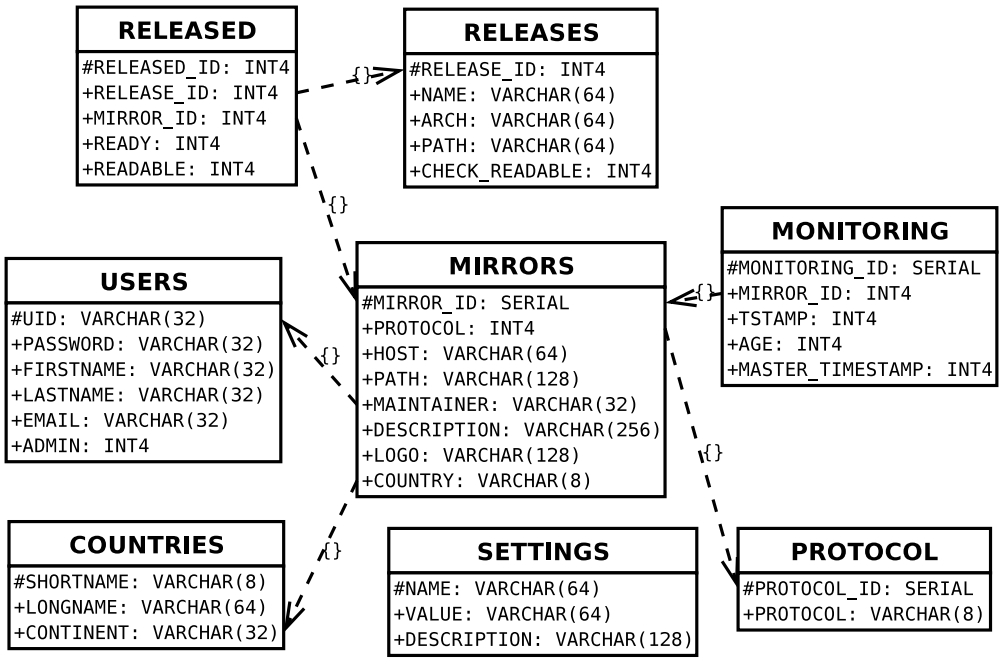


Figure 3-2. Database Design

The data-type SERIAL is used in many tables as the primary key. The field with this data-type is never updated or inserted into the database but automatically incremented each time a new tuple is written to that table. In this way, it is ensured that the primary key stays unique as it is required for a primary key.

- **USERS** - This is the table containing the information about the users allowed to access this system.

- **PROTOCOL** - This table will be used to store the protocols the *mirror servers* provide. The content of this table will be probably be something like http, ftp and rsync.
- **MIRRORS** - This is the table where the data about each *mirror server* are stored. If a mirror has a maintainer, it will also be referenced in this table. Each maintainer can be responsible for multiple mirrors. The maintainer field is a *foreign key* referencing the USERS table. The protocol field is also a *foreign key* referencing the PROTOCOL table. This table was enhanced to include a description for the *mirror server*, a link to a logo to be displayed next to the mirror and a field for a country code in which this *mirror servers* is located. These enhancements were not present during the Interim Report.
- **SETTINGS** - This table is only used to store settings for the front-ends using this database. Possible entries are something like the e-mail address the system is using to send mails or the boundary values used to decide about the state of the *mirror server*. These boundary values are discussed in Section 3.3.1. This table was designed with the goal of offering flexibility. This goal is reached by only offering simple name-value pairs without hard-coding any configuration settings into the database design. In this way, it is simple to enhance the whole system without needing to change the layout of the database.

  This table is the only table without any contact to the other tables in the database. This is due to the fact that all the other tables store application data and this table stores data about the application.
- **MONITORING** - This table was created after the Interim Report was finished and therefore was not been mentioned there. This is the table where results of the mirror assessment are stored like it is discussed in Section 3.3.1.

  The database field MIRROR_ID is again a *foreign key* referencing the corresponding field in the table MIRRORS. As this is the table containing the essential

data for the mirror assessment, it is important to known the details of this table. The field TSTAMP contains the time-stamp of the time at which the measurement represented by each tuple was made. The next field AGE contains the age of the *mirror servers* representing the last update of the data on that *mirror server*. The last field MASTER_TIMESTAMP contains the value of the time-stamp from the master server. This field is also used to determine which measurements were made at the same time.

- **RELEASES** - This table was also created after the Interim Report was finished. This table is used to store the information about what releases are available (NAME), for what architecture this release is (ARCH), where it can be found (PATH) and if it should be checked for the correct permissions at all.
- **RELEASED** - This table is used to store the information about a release in connection with the *mirror server*. The field RELEASE_ID is a foreign key to the RELEASES table and the field MIRROR_ID is used to establish a connection which *mirror server* this information concerns. The READY field is used to indicate if this *mirror server* is ready for this release and the field READABLE shows if the *mirror server* is open to the public.
- **COUNTRIES** - This table is used to establish a connection between the country a *mirror server* is located and to which continent this country belongs. This data are used when creating a list of available *mirror servers* to group these servers accordingly to the continent they are situated which offers the user an easy way to select a *mirror server* which is the closest.

3.3. Mirror Assessment

The mirror assessment consists of two parts. The first part is the check whether the content of the mirror is regularly updated and the other part is which mirror is the

closest, in terms of network distance. As these two parts of the mirror assessment are rather different they are handled completely independently.

In this section, the words *master server*, *monitoring server*, *assessment server* and *mirror server* are used quite often. For a better understanding, see Figure 3-1. The term *assessment server* is used as a synonym for *monitoring server*.

3.3.1. up-to-dateness

The first part of the mirror assessment is whether the content of the mirror is up-to-date. The best way to achieve this is to check every file on the mirror against a database of the actual content of the master server. This is, however, not feasible as it would require huge amounts of traffic to be transmitted for every single mirror server that is checked. To achieve a really good check, the complete content would have to be transmitted with every check. Implementing something like this would lead to a situation where the check to see if the files are up-to-date would consume more network traffic than anything else.

To minimise the network traffic the mirror assessment only analyses a single file containing a time-stamp. This time-stamp is created by the master server every ten minutes. By comparing the time stamp on the master server with the time-stamp file on any of the *mirror servers*, it can be easily determined at what time the mirror was last synchronised. The content of the time stamp is created with the command

```
date +%s
```

and contains the number of seconds since *00:00:00 1970-01-01 UTC*. This date is known as the ***Epoch*** and it is when the Unix time system began. This format has two advantages. The first one is that no matter in what time zone the master server or the assessment server is located, the value will always be the same as the Epoch always has the same time zone (UTC). This avoids the possibility of an error due to

different time zones. The other advantage is that this format is easily parsable as it is a common format on many Unix and especially Linux systems.

The disadvantage of the time-stamp file is that a malicious mirror administrator could only modify the time stamp file to pretend that the mirror he manages is up-to-date while he changed the content to contain modified software. A way to circumvent this scenario would be to sign the time-stamp file digitally. This, however, has not been implemented and can be a point on the list of features of the next version.

This part of the mirror assessment can be done from one server located in any part of the Internet as the distance between any *mirror server* and the server doing the checks does, most of time, not matter. It may only matter if a connection between a special *mirror server* and the assessment server is broken. To circumvent this scenario, the assessment of any mirror is not based on one check but depends on multiple checks.

To get a result which does not depend on a single measuring point, the following has been designed and implemented on the assessment server. The assessment of each mirror is never based upon a single value but always depends on multiple measuring points. The number of measuring points is configurable and the default value is seven. This means that before a *mirror server* is marked as not available because it is out of date or unreachable it requires seven measuring points with a negative result (either out of date or unreachable). The reason for not marking a *mirror server* after the first negative test result as unavailable is that tests showed that any good *mirror server* may return one or more negative test results. A good *mirror server* may seem unreachable for of many reasons, but it may for example just be very slow due to high load on the *mirror server*; this, combined with a large distance to the *mirror server*, may result in the same symptoms as if the *mirror server* were down. Therefore, the assessment is always based on multiple measuring points. In

addition to this, the *mirror servers* are assessed on the age of their content, which is again only based on the time-stamp file. With the age of each mirror, an additional assessment is made using two boundary values (“Old_Age” and “Kill_Age”). These two values are used to mark the *mirror-server* list. The first value (“Old_Age”) is used to mark the mirror as not totally up-to-date. The following is an example of one of the possible clients using the result of the mirror assessment:

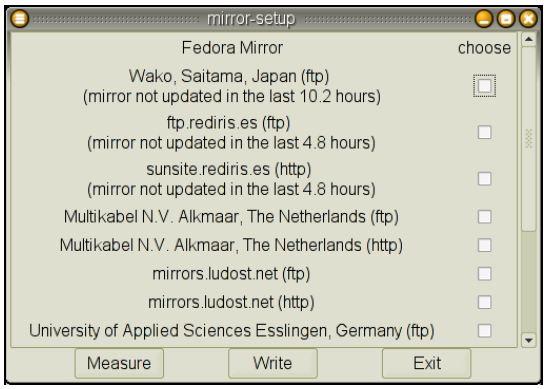


Figure 3-3. Mirror Setup GUI

In this example, it can be seen that some *mirror servers* are marked with a remark like *mirror not updated in the last 10.2 hours*. This is the mark produced by the first boundary value. In this example, this value is set rather low to demonstrate the effect of this first boundary value. For a mirror to be marked as outdated, it is, however, not necessary that multiple measuring points returned that a *mirror server* has outdated content, but only the value of the last measuring point is used.

The effect of the second boundary value (“Kill_Age”) cannot be seen in this example, because the second value is used to remove *mirror servers* from the list with outdated content. In this way, outdated *mirror servers* are not even offered as a possible selection. Another view, however, demonstrates the usage of both boundary

values. The following is an example of the *mirror-server* status web page:

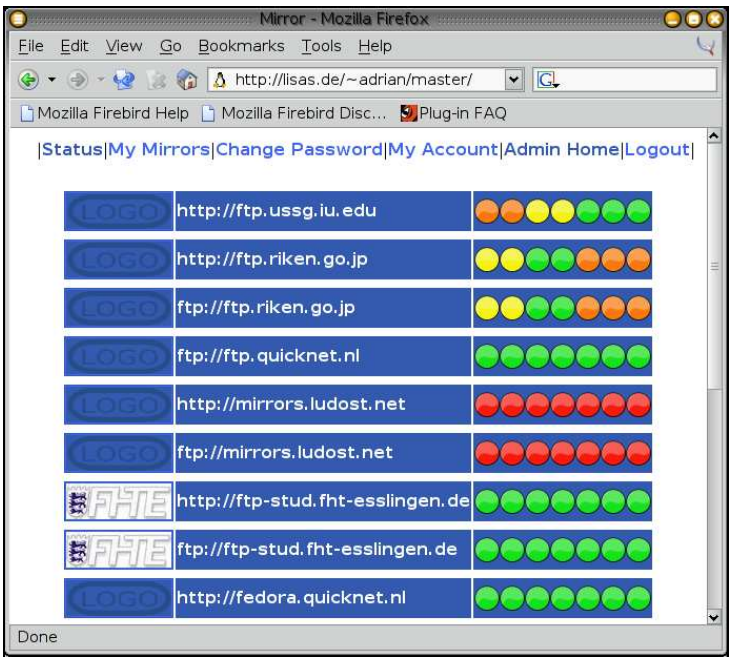


Figure 3-4. Mirror Status Page

In this example, a green dot represents a measuring point with a good return value and the age of the mirror content is within both boundary values. A yellow dot shows that the first boundary value has been reached and from now on the mirror will be marked in the mirror list as not totally up-to-date. An orange dot means that the second boundary value has been reached and this mirror will from now on not be available as a possible mirror in any output presented to the users. A red dot means that an error occurred during the check of this *mirror server*. An error can be, for example, that the *mirror server* is down, does not answer due to high load or the time-stamp file could not be found.

The check on how up-to-date the content of each mirror is needs to be performed

at regular intervals. As this is a service which is already implemented by the cron daemon[1] it is not necessary for it to be reimplemented. Therefore, the mirror check is just a script which checks all mirrors once, writes the results and then exits. To have it executed at regular intervals by the cron daemon, the following has to be entered in the cron-tab[2]:

```
2 * * * * /path/to/checkmirrors.pl
```

3.3.2. Best Mirror

The second part of the mirror assessment is which *mirror server* is the best for each user. This part, however, cannot be done from the assessment server but has to be done from the client side. If it were done on the assessment server, the result would not be representative for any client on the Internet but only for clients located in the same part of the Internet as the assessment server. A possible design which could handle this would require many assessment servers all around the world and it would still not offer the best results for all clients using this service. Therefore, the current design to find the best *mirror server* from the client side makes the most sense.

The optimal solution for the client to determine the best *mirror server* would be to test for the following three measurements:

- bandwidth
- network distance (hops)
- *ICMP* echo reply time.

As this dissertation is about *Free Software*, it was logical to use existing *Free Software* tools to accomplish these measurements. Two tools were quickly

1. cron - daemon to execute scheduled commands
2. cron-tab - table containing the configuration which is driving the cron daemon

found which offered the requested functionality.

To measure the available bandwidth, the programme *bing*[7] was examined. *bing* determines the real throughput on a link by measuring ICMP echo requests round-trip times for different packet sizes for each end of the link. Although this sounds very good in theory, the tool did not offer the expected functionality. The first obstacle was that it requires not only the destination host but also the network device before the destination host on the network route from the source host to the destination host. Although it requires more effort to use this tool, it is still feasible to do so. It requires just an additional step in which the network route is determined. The next obstacle, however, cannot be circumvented as, in order to function properly *bing* properly that symmetric network routes are used. Asymmetric network routes are something which this tool cannot handle. As asymmetric routing is something which is often used to reduce costs, it is not uncommon. The problem is that it is not trivial to detect that an asymmetric routing is used between source and destination host from just one side of the network connection. And as this is the part that can only be done from the client, the network route from the destination to the source system is not available to the client. So, as asymmetric routing cannot be detected, this would render the measurement of the available bandwidth with *bing* unusable as an error during measuring could either be a real error between client and server or just network topology causing this tool to fail. Another problem with *bing* was that, even if symmetric routing was used, the results were never even close to the actual values the network link would support. All these problems led to the decision not to use *bing* to measure the bandwidth.

As the tool to measure the highest available bandwidth between client and *mirror server* did not offer the functionality required, another tool was necessary. The next tool investigated was *netselect* [8]. *netselect* is also based on the analysis of *ICMP* messages transmitted from the client to the server. *netselect*, however, does not try to measure the available bandwidth but tries rather to determine the

best server by analysing the round-trip time of *ICMP* messages as well as the network distance between client and server. *netselect* then selects the server with the lowest *ICMP* round-trip times and the shortest network distance as the best server to connect to.

netselect proved to be quite reliable in the tests undertaken and the results were realistic enough to be used in the final programme running on the client side.

3.4. Security

This section deals with possible security-related problems which were identified during the design phase of this dissertation.

3.4.1. Password Handling

The password required to authenticate each user is one of the central points of the security mechanisms used by the web front-end. To secure this “user-name”-password combination against eavesdropping, one of the requirements of this project is that the connection to the web front-end has to be a secure connection. The secure connection, however, is not offered as a functionality from this project but as a feature provided by the Apache *HTTP* Server. One of the modules available for the Apache *HTTP* Server offers an encrypted connection between the user’s browser and the web server by implementing *HTTPS*

As all parts of the dissertation are based on open and freely-available software, it was obvious that the encryption of the connection between user and server would be based on an already existing project. The Apache *HTTP* Server and the module responsible for the *HTTPS* are widely deployed and therefore offer much higher and

better security than any new implementation of the *HTTPS*. It is also more likely that an error in the current implementation will be found and fixed much sooner. It also avoids unnecessary complexity in this dissertation as the security of the connection is just a service required and not a focal point of this dissertation.

The security of the password transmission from the user to the web server is now protected. Additional security for the password is now offered as the plain text password is not saved in the database but is converted by using a one way hash function. The generated hash is exactly 31 characters long and written into the database. This protects the password to be retrieved from the database even if the system is compromised. As it is a one-way hash, it should not be possible to extract the plain-text passwords except by a brute force attack on the password hashes. Even for user authentication, the plain-text password is not necessary. The user enters the plain-text password in his browser and this password is then transmitted over the secure *HTTPS* connection to the server. The server then generates the *MD5* hash from the password and compares the hash against the hashed password in the database.

As nobody is able to read the passwords from the database, it is also not possible for the person administering the system to read a password. This means that, if a user forgets his password, the account can only be reset with a new password.

3.4.2. HTTP Referrer

The *HTTP* Referrer is a header transmitted by a browser to a web server. This header contains the *URL* of the web page visited last. It might be important to know that the name of the *HTTP* header is misspelled and that the actual *HTTP* header transmitted looks like this:

```
Referer: http://lisas.de/kover/
```

This header will be used to check and ensure that the input from the *SQL* commands

executed by the *PHP* scripts comes from one of our own *PHP* scripts. This check can be easily performed with *PHP* using the global variable

```
$_SERVER[HTTP_REFERER]
```

which contains the Referer [sic] header. So just by a comparison of the *HTTP* Referrer transmitted by the client and the expected Referrer, this threat can be minimised. Even if the *HTTP* Referrer is not checked, this would be not a big problem, because this check only guarantees that the parameters passed from one *PHP* script to the next are actually the parameters the scripts are designed to handle. This means that values passed from text fields do not present any threat as the user can enter any text he wants. The only area where this could be exploited are lists or combo-boxes where the user has to make a selection from a predefined list of possible values. An example can be seen in the following screen-shot where it is possible to select which protocol the mirror supports:

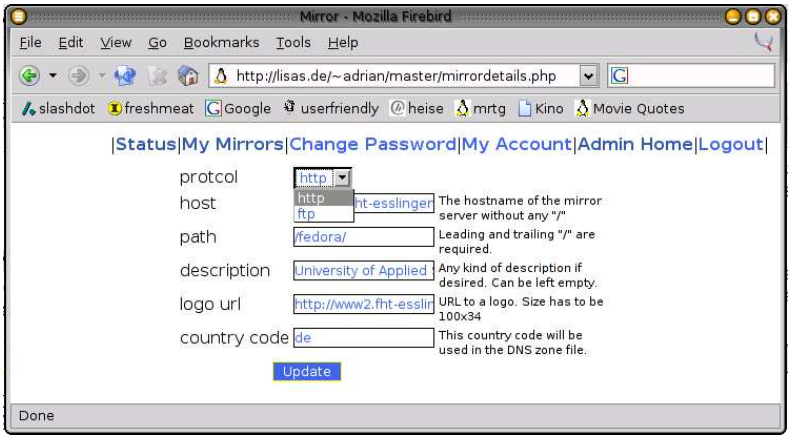


Figure 3-5. Protocol Selection

This would be a case where a malicious user could enter other values than the

two available protocols in this example by locally modifying the web-page. He could then transmit the value he wants in the database to the script doing the *SQL* operations on the server. It is important that he cannot execute any *SQL* command he wants to. He does not have the possibility of injecting any *SQL* command into the system if the referrer is not checked. It is only possible to enter any value even if it is designed to transmit only values from a predefined list. By not checking the referrer it is not possible to execute arbitrary *SQL* commands. The threat of *SQL* injections is the topic of the next section.

Although the *HTTP* Referrer can be verified, this is, however, by no means a way of protecting the system against malicious users trying to exploit the system. The real problem is that the *HTTP* Referrer header cannot be trusted as it is something transmitted by the client using the system (browser). The usual behaviour of the standard web browsers is to transmit the *HTTP* Referrer to the system but this could easily be changed either by using a special browser which allows tampering with the transmitted Referrer header or even using some special tool designed for a task like this. But as already mentioned, this does not pose a security threat, it is just something to be aware of.

An easy solution to arbitrary text being entered at points where is should be selected from predefined lists is to check whether the input transmitted from the client is still one of the possible solutions which have been offered to the user in the first place.

3.4.3. SQL Injection

SQL Injections are a common exploit with many database-backed websites offering a way for user to change or enter data. Instead of the text he is expected to enter the malicious user enters an *SQL* command to be executed by the database. This enables this user to execute any *SQL* command he wants to with all the rights the web front-end has on the database. This is one of the reasons why the web front-end should

only have the rights to modify the database which are really necessary. But even if the access rights only offer the necessary functions for the web front-end, it might not be possible for the attacker to delete or create any tables or databases but he will still be able to enter or delete any data he wants. This cannot be avoided as at least some rights are necessary to access or modify the data from the database; without these rights, a web front-end for a database makes no sense if the data cannot be used.

A good way to avoid attacks with *SQL* Injections is to validate all the input from the user. An easy step is to check every field to see whether the data type corresponds with the format of the database tables. This is not only a security precaution but also a way to circumvent *SQL* errors triggered by wrong data types. With checks like these, the only way to perform a *SQL* Injection is by modifying a string. And this threat can easily be mitigated by analysing the input string to see whether it contains any *SQL* commands. After this analysis, the whole string can either be discarded or just changed so that all the *SQL* commands are removed.

Chapter 4. Results

What do they think I am, an engineer?

Calvin

After the design of all the relevant parts of the dissertation was finished, the following results were obtained. This chapter deals with the programs and programming languages used and the results to which they lead.

4.1. Programmes and Programming Languages Used

This section offers a short introduction to the programs and programming languages used and why they were used.

4.1.1. Programmes

- **Apache HTTP Server**[9] - “The Apache *HTTP* Server Project is an effort to develop and maintain an open-source *HTTP* server for modern operating systems including UNIX and Windows NT. ”[9]

  A *HTTP* server is necessary at many points of this dissertation. The first and most important part where the *HTTP* server is required is the web front-end. Without an *HTTP* server, it would not be possible to offer an interface like that which is one of the central points of the whole system. The second part where the *HTTP* server is required is described in Section 3.3.1. The Apache *HTTP* Server was chosen because it has been the “the most popular web server on the Internet since

April of 1996” [9] and because it was the *HTTP* Server available on the server I have access to.

- **PostgreSQL**[10] - “The PostgreSQL Global Development Group is a community of companies and people co-operating to drive the development of PostgreSQL, the worlds most advanced Open Source database software.”[10]

  This is the *SQL* database used as described in Section 3.1.1. There were two possible choices of *SQL* databases. PostgreSQL was chosen because it was the system I was most familiar with.

- **netselect**[8] - “[...]an ultrafast intelligent parallelising binary-search implementation of “ping.”” [8]

  The reasons why this tool is used are described in Section 3.3.2.

4.1.2. Programming Languages

The following programming languages were used to write the necessary source code for this dissertation.

- **PHP** - “a general-purpose scripting language that is especially suited for Web development and can be embedded into *HTML*. ” [6]

  As already mentioned, *PHP* is used to develop the web interface with which the *mirror server* administrators will interact and will provide anybody with a graphical representation of the current status of registered mirrors.

- **Perl** - “is a stable, cross platform programming language. It is used for mission critical projects in the public and private sectors and is widely used to program web applications of all needs.” [11]

  Although *Perl* is advertised as a programming language used to write web applications, it is used for another purpose in this dissertation. *Perl* is the language

used in the part which does the monitoring of the *mirror servers*.

- **Python** - “is an interpreted, interactive, object-oriented programming language.”[12]

  Python is used to write the application available to the end user which helps to select the best available mirror. In this case, not only *Python* is used but also the PyGTK[13] wrapper which provides access to the *GTK+* library from *Python*. With PyGTK it was easy to provide the end user with a *GUI*.

4.2. Tools written

This section is about the programmes developed for this dissertation. The decision which programming language to use and a short description will be provided for each tool. All code written during this dissertation is released under the *GPL*.

4.2.1. checkmirrors.pl

This is the tool which monitors all registered *mirror servers*. This is the part which is written in *Perl*.

The reason for writing this tool in *Perl* is a rather simple and personal one. As this tool provides one the central functions in the whole system, it was important that it was available at a very early stage of the development. As I was already familiar with programming *Perl* and also already knew how to use the required modules (*DBI*, *HTTP*), *Perl* was the way to go.

Although *Perl* offers the option to develop in a object-oriented way this tool was developed as a purely procedural programme. The reason for this is that this tool will never be very big and, for the task it is designed to do, the object-oriented approach would only have added overhead during development and runtime.

The following is a flowchart of the `checkmirrors.pl`-script:

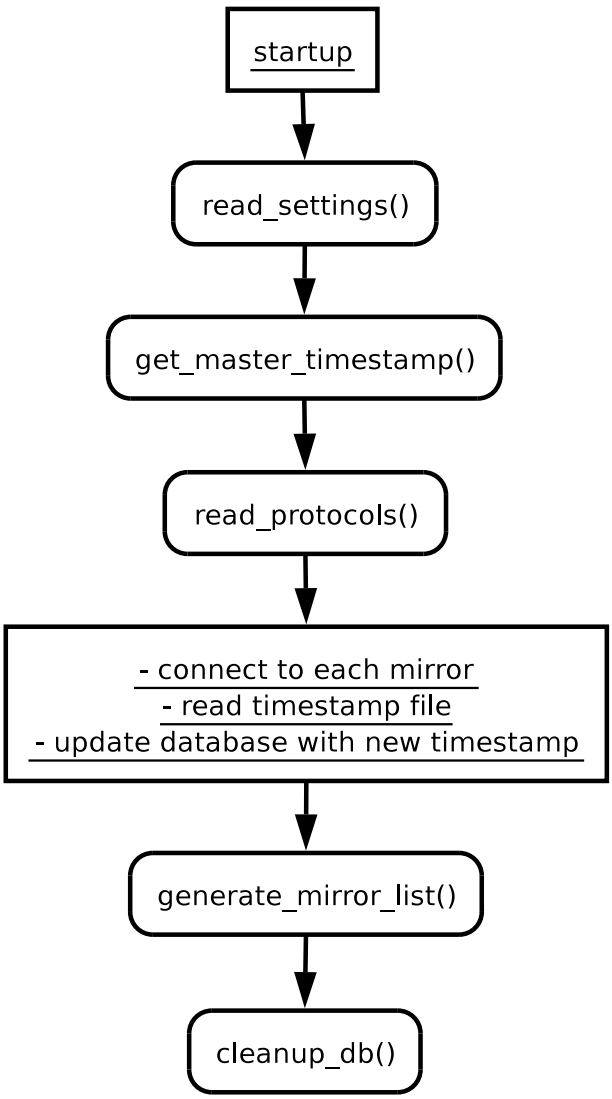


Figure 4-1. `checkmirrors.pl` flowchart

As can be seen in the flowchart, there are no forks in the flow of this programme as it has a very simple design due to the simple and straightforward task it has to do.

- `startup` - during startup, the command-line arguments are parsed and some

variables and the database connection are initialised. If a command-line argument is passed on to the application, it will run in debug mode, meaning that it will print out what it is currently doing. The variables initialised define the names of the different output files, the database name and the days after which entries should be removed from the database.

- `read_settings()` - this is the next function called. As the name says it simply reads the settings from the database. These are the settings which can be configured with the web interface.
- `get_master_timestamp()` - again the name of the function tells it all. To be able to determine the age of the content of each of the registered *mirror servers* it is necessary to have the time-stamp of the master server. Therefore, the address of the master server is read from the database and then the `TIMESTAMP` file is retrieved from the master via *HTTP* just like it will be done with all the registered *mirror servers.*
- `read_protocols()` - this function reads all supported protocols by the system from the database and fills an associative array of scalars (hash). This hash is only used to read the available protocols.
- `connect to each mirror` - a connection to each *mirror server* is made. The protocol used for these connections is the one specified in the database. The connection to each mirror is done sequentially in a loop. This means that, if one *mirror server* is not responding, the programme will do nothing until a timeout occurs. The timeout is specified as 30 seconds and can be changed if required. Tests were made with a smaller value for the timeout, but that resulted in some *mirror servers* being rated as unreachable although they could only not be reached due to a combination of high load on the server and large network distance between assessment server and *mirror server*. Therefore, it was decided to use a rather long timeout. If the number of *mirror servers* increases, this timeout

can become a problem if many *mirror servers* are not available and the timeout is reached. In a worst-case scenario, it might even happen that the runtime of this programme exceeds the interval in which this programme is started. A possible solution for this problem would be to connect all or at least some *mirror servers* in parallel.

`read timestamp file` - after a connection has been successfully established, the file `TIMESTAMP` is retrieved and evaluated. Using this value, the age of the mirror is computed using the age of the master server as reference. The age and all other details of the mirror are written to another hash.

`update database with new time-stamp` - the just computed age of each *mirror server* is now written to the database. These values are used in the web front-end to display the status of each *mirror server*.

- `generate_mirror_list()` - this is the function which generates the various output files. The function reads the just written values from the database including a certain number of measuring points from previous measurements. The number of measuring points read from the database depends on a setting which has been extracted from the database during `read_settings()` at the beginning of this programme. This means that if the value is changed while this programme is active, this change will not be realised before a restart of this programme.

  The output files written may not include all registered mirrors. The number of mirrors included depends on the age of the *mirror servers*. As already mentioned above (Section 3.3.1), there are two boundary values used during the assessment of the *mirror servers* These boundary values also apply here and may therefore remove some *mirror servers* from the output file or a remark will be added depending which boundary value was reached.

  After the data are read from the database they are written to various output files. To offer the possibility to write the data to many output files with different for-

mats, the interface to write the files was made as flexible as possible. A function with the name `write_generic()` was introduced which just calls any function writing a specific output format and thus introducing the ability to create different output plugins. The here called plugins are the point where the boundary values from above (Section 3.3.1) are finally really evaluated. Up to this point, the boundary values were just a parameter passed on by every function. Using this technique the current implementation offers the following two output formats:

The first output format can be used as an include file for the *BIND* name server. To include the output in one of the zone files from *BIND*, the following statement is necessary:

```
$INCLUDE db.mirrors
```

The current implementation writes the content in the format of a *BIND* zone file named `db.mirrors`. An example of the content is:

```
; autogenerated by checkmirrors.pl
; Mon May 17 15:02:15 2004

ftp.us  A  129.79.5.130
      TXT  “Indiana University, Bloomington Indiana, USA”
ftp.es  A  130.206.1.5
      TXT  “sunsite.rediris.es”
ftp.de  A  129.143.116.10
      TXT  “University of Applied Sciences Esslingen, Germany”
ftp.us  A  128.171.104.133
      TXT  “University of Hawaii Honolulu, Hawaii, USA”
ftp.nl  A  213.73.255.253
      TXT  “Multikabel N.V. Alkmaar, The Netherlands”
ftp.ro  A  192.129.4.120
      TXT  “Romanian Education Network, Iasi, Romania”
ftp.jp  A  134.160.38.142
```

```
      TXT  “Wako, Saitama, Japan”
ftp.us  A  204.152.189.120
      TXT  “Kernel.org, San Francisco California, USA”
ftp     A  130.206.1.5
      TXT  “ftp.rediris.es”
ftp.nl  A  213.73.255.253
      TXT     “Multikabel N.V. Alkmaar, The Netherlands”
```

If this file is included in the zone file for the domain *example.com*, it adds the *IP*-addresses from above to that domain. It is now possible to query the name-server for the address of the ftp-server *ftp.<countrycode>.example.com*. This offers the user the possibility to enter this into a configuration file without needing the exact name of the *mirror server*. If there are more servers for one country, the name-server will return a different address for each query and thus offer a simple kind of load-balancing between the available ftp-servers. An additional feature with this zone file is the possibility to query the *TXT* entry to get more information about each *mirror server*.

```
$ host -t TXT ftp.jp.example.com
ftp.jp.example.com text “Wako, Saitama, Japan”
```

The first boundary value concerning the age of the *mirror servers* is not evaluated during the writing of the zone files. If the age of a *mirror server* is greater than the “Old_Age” it will not change anything and if the age is greater than the “Kill_Age” the *mirror server* will not be mentioned in the output format.

The second output format is used to create a list of mirrors which can be used with the `mirror-select.`*lua* script which is part of *apt*. This output plugin is simpler than the previous one and the following is an excerpt of the output file:

```
Wako, Saitama, Japan (http) \
(mirror not updated in the last 11.0 hours)
http://ftp.riken.go.jp/pub/Linux/\
```

```
fedora.us/fedora fedora/1/i386 stable
--
sunsite.rediris.es (http)
http://sunsite.rediris.es/mirror/\
fedora.us/fedora fedora/1/i386 stable
--
Multikabel N.V. Alkmaar, The Netherlands (http)
http://fedora.quicknet.nl/fedora fedora/1/i386 stable
```

This output shows the effect of the first boundary value ("Old_Age"). The comment "mirror not updated in the last 11.0 hours" is created if this boundary value has been crossed.

Both implemented output plugins are very specific concerning the format of the files written. It would have been possible to write an additional output plug-in which would create rather generic files which could then be parsed by any application requiring it. For such a generic output plug-in *XML* would probably have been a very good choice. Such an output plug-in, however, was not implemented as there was no application requiring it.

- `cleanup_db()` - this function was introduced later in the development of the system. After about two months, there was already so much data in the database that it became necessary to remove old and unneeded data from the database. This task is done by the function `cleanup_db()` which runs every time this programme is executed and removes all entries from the database which are older than 7 days.

This programme is very static in matters of configuration. Every configuration option can only be changed by editing the corresponding variable in the source code. It would have been possible to make it all configurable via command-line arguments but as it is currently only run on the assessment server and nowhere else, there was no need to make it configurable. If it is ever necessary to run it on another server, it

will be very easy to edit the script so that these variables hold different values. The decision to have to edit the source code to change the value of a variable was made because this feature is currently not necessary and the programme is usually only run automated in the background. So it only needs to be configured once and not every time it is run.

4.2.2. mirror-setup.py

This is the programme which should be used by the end-user to help to decide which *mirror server* offers the best service for this user. For details on how this is accomplished, see Section 3.3.2. The short version is that only mirrors which are up-to-date are presented to the user and, with the help of the tool *netselect*, the fastest and closest *mirror server*, is then selected.

This part of the system is written using the programming language *Python*. There were two reasons for using *Python*. The first is a very simple reason. I personally always wanted to work with *Python* and this part of the dissertation offered a good opportunity to learn *Python* using the *GTK+* bindings through *PyGTK*. As I already had experience programming *GUI*s using *GTK+*, this combination was the optimal solution. The second reason for using *Python* was that *GUI*-programming is much easier using an object-oriented language such as *Python*.

The following is the class diagram of the current implementation of this programme:

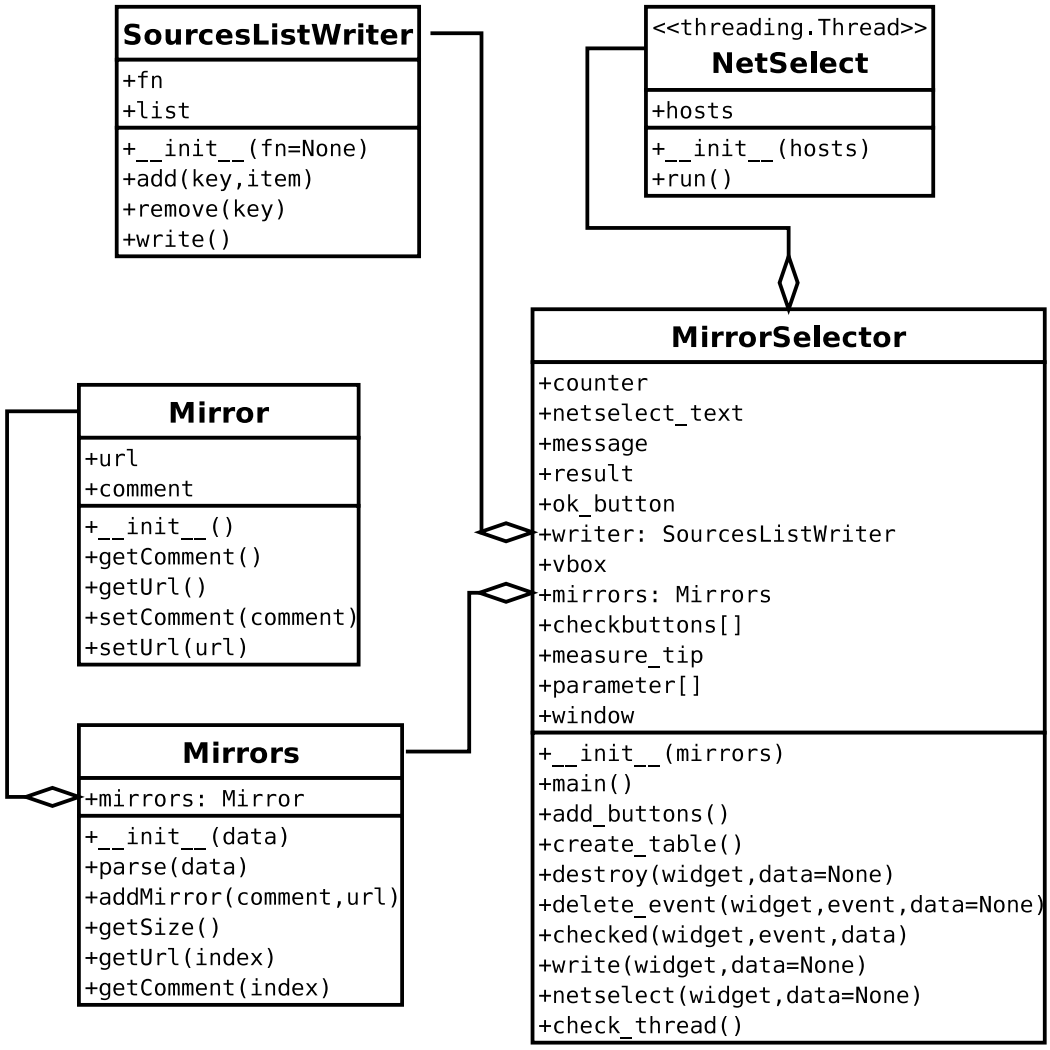


Figure 4-2. mirror-setup.py Classes

The main class is the `MirrorSelector` class. This is the first class instantiated by the programme after the initialisation and command-line parameter parsing. In this class, the complete *GUI* and instances of the used classes (`Mirrors` and `SourcesListWriter`) are created. Figure 3-3 shows a screen-shot of the *GUI*.

If the button Measure is pressed, an instance of the class `NetSelect`, is created which is used to start the tool *netselect* in a thread to determine which *mirror server*

is the closest and fastest. The reason for using an extra thread to run *netselect* is that the *GUI* stays responsive to input from the user and also offers an output that the measurement is still running. The input for *netselect* is the addresses of the *mirror servers* which are stored in the class `Mirrors`. The input for this class is the content of the file created in Section 4.2.1. This file is downloaded and parsed and each mirror server entry is stored in the class `Mirror`. These classes are stored in a linked list which is managed by the class `Mirrors`.

If one or more *mirror servers* have been selected, either by the user or with the Measure button, a `sources.list` compatible file can be written using the class `SourcesListWriter`. This file can then be used by *apt-get* to download software from the selected *mirror servers*.

4.2.3. Web Interface

This is the part written using the programming language *PHP*. The reasons for this decision are simple. This part of the dissertation had to be finished at an early stage as it is also a very central part of the whole system as well as the part discussed in Section 4.2.1. As *PHP* is a programming language with which I am very familiar, this was again a decision based on personal reasons. Nevertheless, it was the right choice. *PHP* is designed to handle tasks like these and makes programming web interfaces very easy and straightforward and therefore it was definitely the correct decision.

The web interface has different functionalities and not just one like the two previous parts (Section 4.2.1 and Section 4.2.2). One functionality is to display graphically the status of all *mirror servers* which can be seen in the screen-shot Figure 3-4. In this screen-shot, the last 7 measuring points are displayed. The number of displayed measuring points is configurable and can be adjusted to meet the requirements of any project. The meaning of the colours representing the measuring points is dis-

cussed in Section 3.3.1. This is the only functionality available without the need to log in.

If the user has successfully logged in, he has the possibility to view the mirrors this user is responsible for (My Mirrors), change the password (Change Password), edit the details of the logged-in user (My Account) and of course logout again (Logout).

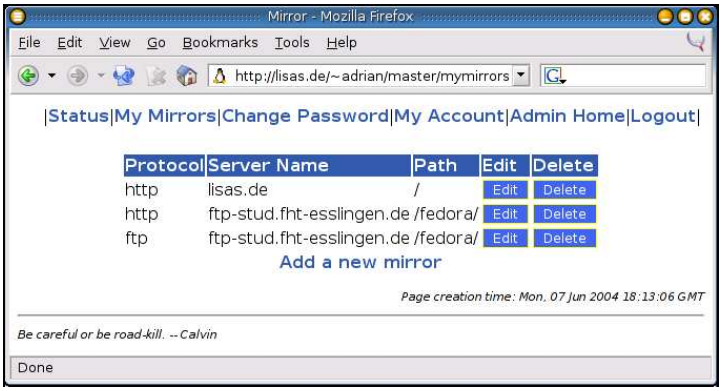


Figure 4-3. My Mirrors

Figure 4-3 shows a screen-shot of the page if My Mirrors is selected. The user has the possibility to manage the details of his mirrors or add a new *mirror server*. If the button Edit is selected, the name and logo displayed on the start page can be changed as well as the country code used in the *DNS* as described in Section 4.2.1.

An additional button (Admin Home) is available when the logged-in user has administration rights. These rights mean the user can add a new user to the system (User Administration), manage the *mirror servers* (Mirror Administration), and change certain values (Global Settings) like the boundary values from Section 3.3.1 or the number of probes used for the mirror assessment. In the

section Mirror Administration it is not only possible to add new *mirror servers* but it is also possible to assign a user available in the database as maintainer.

4.2.4. checkmirrorpermission.pl

When most of this dissertation was already finished one more requirement was detected. To offer not only the possibility to monitor the up-to-dateness of the mirrored data but to also offer the possibility to check if a certain release mirrored is already available to the public.

The reason for the requirement of this feature is the fact that a release of a software product in the *Free Software* community follows, most of time, certain rules. The following description is an example how the distribution process over the Internet is handled by Red Hat. A distribution is usually released worldwide on a certain date and time. To still use the advantage of distributing the data via a system of *mirror servers* the data have to be available on all *mirror servers* at exact the release date. Without being available at the release date the *mirror servers* are almost of no real use as everybody wants to download as fast and early as possible.

The last released distribution from Red Hat (Fedora Core 2) was almost 30GB big and even with the fastest network connection it requires some time to sync the *mirror servers* with Red Hat's master servers. Therefore it is necessary that Red Hat distributes the necessary data before the release date to the *mirror servers*. The *mirror servers* on the other hand have to be careful that this data is not released to the public before the release date.

To get an overview which *mirror server* has already changed the permissions on the release so that everybody can download it, no matter if the permissions have been changed by accident or by purpose, an additional programme has been written. This programme is called `checkmirrorpermission.pl`.

The `checkmirrorpermission.pl` programme is written in *Perl* and has many similarities with the `checkmirrors.pl` programme described in Section 4.2.1. The difference between these two programmes is that the `checkmirrorpermission.pl` programme only connects to the *mirror servers* and then updates their status in the database depending on the permissions of the directory containing the release without writing any files. All actions depending on the result of this task have to be performed in another part, if that is required or desired.

The following is a flowchart of the `checkmirrorpermission.pl`-programme:

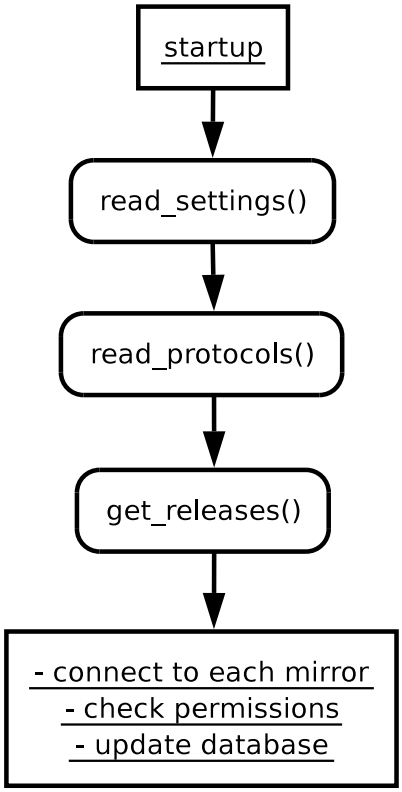


Figure 4-4. `checkmirrorpermission.pl` flowchart

As this programme is based on the source code of the `checkmirrors.pl`-programme the flowchart is similar to the flowchart presented in Section 4.2.1. The startup and the data read from the database is done in the same sequence as described in Section 4.2.1. The only difference is in the part where the actual task is performed. Instead of connecting to all the *mirror servers* an reading the `TIMESTAMP` file the programme tries to read the directory containing the release which is currently active. The result of this operation is then stored in the database.

Before a certain version is officially released to the public all *mirror servers* should deny the access to the directory containing this release. If a *mirror server* is open to the public before the release date it is possible to take appropriate actions to change the permissions of the directory containing this release.

Chapter 5. Conclusions

No one can prove I did that.

Calvin

5.1. Overall Aims

If the results are compared with the aims stated in Section 1.2, I rate this dissertation as a success. All three aims have been fulfilled. The first aim to develop a system using only *Free Software* was definitely reached as all tools and programmes used during the whole development process were *Free Software*. The second part of the first goal, to use the help of the *Free Software* community, was not practised very eagerly. There were, of course, discussions with some people but these discussion never led to any big changes concerning the aims to be accomplished in this dissertation.

The second aim was to create a system supporting the *Free Software* community. Whether this goal has been fulfilled is difficult to evaluate. From my point of view, the goal has been reached. From the point of the community, it cannot yet be said. The usefulness of this project for the community can only be evaluated if it is widely used. The project will not support the community if it is not used. To achieve a wide deployment of this system, it has to be announced to the public and, if the community finds it useful, it will be adopted by different people. On the other hand, however, it will not be used if nobody likes it. The fulfilment of this goal is a problem which I will address after this dissertation is finished and the evaluation is therefore beyond this dissertation.

The third and last aim, to use existing expertise as well as learn new techniques, was the easiest goal to fulfil and was of course reached. A good example of fulfilling this

goal is the programming required to implement the different parts of this project. Some parts were created using programming languages I was familiar with (*Perl*, *PHP*), and other parts were created with programming languages that were new to me (*Python*).

5.2. Implementation Specific Goals

The aims and objectives formulated during Section 1.2 have proved to be quite reasonable. During the whole time the dissertation was under preparation these aims did not change and could therefore be followed pretty closely. Although the database has changed slightly between the Interim Report and this Dissertation, it has proved to be correct and usable for all the other parts accessing it. A good decision during the design of the database was that the tables have been normalised to reach *Third Normal Form* where possible. Especially as the different parts sometimes requiring very different and special data, the database normalisation was very helpful.

The next aim from Section 1.2 is to provide a mean to monitor the status of the registered *mirror servers*. This goal was also reached. As already mentioned, the monitoring is one of the central parts of the whole system and therefore it was created at the beginning of this project. During the course of the dissertation, there were not many changes in the monitoring, which leads to the conclusion that the design and initial programming was rather good and fitted the requirements. The monitoring is also one of the best-tested parts as it has now been running constantly for more than 4 months without any problems.

The web-front-end can be seen as part of the database and as part of the monitoring as it represents the data of both parts. The web-front-end is rather simple. It reads configuration data from the database, reads the application data (information

about the mirrors, results from the monitoring) and displays this application data according to the configuration values just read from the database. Due to its simple character, there is not much to say about it in conclusion. It just worked as expected.

The *GUI* created to be used on the side of the user is unfortunately not very well tested. It worked without problems in the tests I did but there might, of course, still be problems I did not encounter. An important step to moving this client-side application from prototype state to production ready is to announce it to the public and hope that it will be accepted. It is important for any *Free Software* project that the numerous people in the community are aware of the problems the project tries to solve. If the project does not only formulate the problem many are aware of, but even provides a first implementation, the chances are great that this prototype will be used as a point from which the development can start. This might lead to widely used and tested solutions based on the current prototype. The main point for the success of this project is that the problem I see is the same problem many other people also see.

Two of the aims formulated in this dissertation were not reached. The first is the digitally signed time-stamp file. This was not implemented for practical reasons. As the time-stamp file is generated automatically every ten minutes, it requires an automated signing of the file. This, however, is not possible as a useful signing requires a key-phrase to be entered every time the file signed. This makes is difficult to automate the process as it is required and therefore the signing has not been implemented. The second goal not reached is the availability of a command-line interface to administer the system. The command-line interface would have been a nice feature, but the decision was made that this was not a necessary requirement for the dissertation and was therefore dropped. Of course, there is no objection to the implementation of such an interface at any later time.

5.3. Limitations

Although I am very satisfied with the results of this dissertation, there are also many limitations in the current implementation. The method for determining the best mirror at the side of user is not optimal. Although network distance and the time between an *ICMP* echo request and *ICMP* echo reply are measured, this may in some cases not be enough to determine the best mirror for each user. The current implementation, for example, does not take into account that the *mirror server* might be not accepting any connections due to very high load. The reason why the current result is regarded as good enough is that the main goal of this dissertation was to provide a complete system. Therefore, it was decided that is more important to focus on the whole system.

Another shortcoming of the current implementation is that the results from the test to determine the best *mirror server* were not compared with actual bandwidth measurements to create a correlation between the results from *netselect* and the actual available bandwidth. The reason for the lack of tests concerning this part are very simple. I had access to only one computer connected to the Internet via *DSL* and the three best *mirror servers* according to *netselect* were able to fully saturate my connection. Thus I was not able to assess the result from *netselect* in connection with the bandwidth actually available.

Another important thing to remember is the fact that the plan was to offer only a prototypical implementation of all the parts. Although some parts have matured far beyond the prototype stage, some parts still have to be viewed as a prototype.

5.4. Final Conclusion

Although this dissertation was much more work than I had ever expected, it was

great to work on this particular topic as it was exactly something in which I found great fascination. What is even more fascinating is the fact that, after the official dissertation has finished, it might still continue to exist in the public if it is adopted by the community. This is, of course, something which is hard to predict and maybe it will take some time. It also depends on many other factors which I cannot influence. It was great to do something, from my point of view, which offers a solution to a common problem and to work for and with the *Free Software* community. Although the exchange with other people was rather sparse, it was always great to get feedback and ideas from other people.

Glossary

APT - Advanced Package Tool

“ APT features complete installation ordering, multiple source capability and several other unique features[...] ”[1]

BIND - Berkeley Internet Name Daemon

“ BIND (Berkeley Internet Name Domain) is an implementation of the Domain Name System (*DNS*) protocols and provides an openly redistributable reference implementation of the major components of the Domain Name System[...] ”[14]

DBI - Database Interface

DPKG - Package maintenance system for Debian

DSL - Digital Subscriber Line

DNS - Domain Name System

Foreign Key

A foreign key is a field in a relational database table that matches the primary key field of another table.

Free Software

“[...]A program is free software if users have all of these freedoms. Thus, you should be free to redistribute copies, either with or without modifications, either gratis or charging a fee for distribution, to anyone anywhere.[...]”[15]

GnuPG - GNU Privacy Guard

“ GnuPG is a complete and free replacement for PGP.” [16]

GPL - General Public License

“ “GPL” stands for “General Public License”. The most widespread such license is the GNU General Public License, or GNU GPL for short. This can be further shortened to “GPL”, when it is understood that the GNU GPL is the one intended. ”[15]

GTK+ - The GIMP[17] Toolkit

“GTK+ was initially developed for and used by the GIMP, the GNU Image Manipulation Program. Therefore, it is named “The GIMP Toolkit”, so that the origins of the project are remembered. Today GTK+ is used by a large number of applications[...]”[18]

GUI - Graphical User Interface

HTML - Hyper Text Markup Language

HTTP - Hyper Text Transfer Protocol (world wide web protocol)

HTTPS - Hyper Text Transfer Protocol Secure sockets

ICMP - Internet Control Message Protocol

IP - Internet Protocol

LUA

> “Lua is a powerful light-weight programming language designed for extending applications. Lua is also frequently used as a general-purpose, stand-alone language. ”[19]

MD5 - Message Digest 5 - one way hash function

Mirror Server

A mirror server is system which mirrors the data from one server to increase the availability and accessibility of the data; the data on a mirror server are usually not altered and depend on the mirrored data being updated regularly.

Perl

Perl “is a stable, cross platform programming language. It is used for mission critical projects in the public and private sectors ans is widely used to program web applications of all needs.” [11]

PHP - PHP Hypertext Preprocessor (server-side scripting language)

PHP is “a general-purpose scripting language that is especially suited for Web development and can be embedded into *HTML*. ” [6]

Python

Python “is an interpreted, interactive, object-oriented programming language.”[12]

RPM - RPM Package Manager

“ The RPM Package Manager (RPM) is a powerful command line driven pack-

age management system capable of installing, uninstalling, verifying, querying, and updating computer software packages. ”[20]

SQL - Structured Query Language (database query language)

URL - Uniform Resource Locater (world wide web address)

XML - eXtensible Markup Language

References

Internet

[1] *Debian: http://www.debian.org/: (July 2004).*

[2] *Red Hat: http://www.redhat.com/: (July 2004).*

[3] *Conectiva: http://www.conectiva.com/: (July 2004).*

[4] *Fedora: http://www.fedora.us/: (July 2004).*

[5] *Fedora: http://fedora.redhat.com/: (July 2004).*

[6] *PHP: http://www.php.net/: (July 2004).*

[7] *Bing: http://www.freenix.fr/freenix/logiciels/bing.html: (July 2004).*

[8] *netselect: http://www.worldvisions.ca/~apenwarr/netselect/: (July 2004).*

[9] *Apache: http://httpd.apache.org/: (July 2004).*

[10] *PostgreSQL: http://www.postgresql.org/: (July 2004).*

[11] *Perl: http://www.perl.org/: (July 2004).*

[12] *Python: http://www.python.org/: (July 2004).*

[13] *PyGTK: http://www.pygtk.org/: (July 2004).*

[14] *ISC: http://www.isc.org/: (July 2004).*

[15] *The GNU Project: http://www.gnu.org/: (July 2004).*

[16] *Gnupg: http://www.gnupg.org/: (July 2004).*

[17] *GIMP: http://www.gimp.org/: (July 2004).*

[18] *GTK+: http://www.gtk.org/: (July 2004).*

[19] *LUA: http://www.lua.org/: (July 2004).*

[20] *RPM: http://www.rpm.org/: (July 2004).*

Bibliography

Books

[Harlow99] Eric Harlow, 1999, 0-7357-0021-4, New Riders, *Developing Linux Applications: with GTK+ and GDK*.

[Bovet01] Daniel P. Bovet and Marco Cesati, 2001, 0-596-00002-2, O'Reilly, *Understanding the Linux Kernel*.

[McCarty99] Bill McCarty, 1999, 1-56592-705-2, O'Reilly, *Learning Debian GNU/Linux*.

[Wall00] Larry Wall, Tom Christiansen, and Jon Orwant, 2000, 0-596-00027-8, O'Reilly, *Programming Perl: Third Edition*.

[Fuecks03] Harry Fuecks, 2003, 0-9579218-5-3, SitePoint Pty. Ltd., *The PHP Anthology: Volume I: Foundations*.

[Tanenbaum03] Andrew S. Tanenbaum, 2003, 0-13-066102-3, Prentice Hall PTR, *Computer Networks: Fourth Edition*.

[Pressman01] Roger S. Pressman, 2001, 0-07-365578-3, McGraw-Hill, *Software engineering: a practitioner's approach, Fifth Edition*.

[Gamma95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, 1995, 0-201-63361-2, Addison Wesley, *Design Patterns: Elements of Reusable Object-Oriented Software*.

[Coulouris01] George Coulouris, Jean Dollimore, and Tim Kinderberg, 2001, 0201-61918-0, Addison Wesley, *Distributed Systems: Concepts and Design, Third Edition.*

[Schneier96] Bruce Schneier, 1996, 0-471-12845-7, John Wiley & Sons, Inc., *Applied Cryptography: Protocols, Algorithms, and Source Code in C.*

Appendix A. Management of the Project

Verbing weirds language.

Calvin

A.1. Time Management

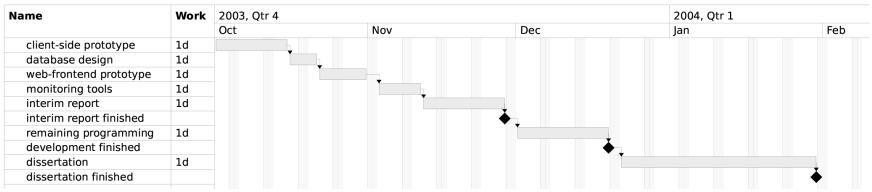


Figure A-1. Gantt Chart during Interim Report

Figure A-1 shows a *Gantt Chart* of the planned tasks and time required to finish this project. This time plan was created during the writing of the *Interim Report*. This time plan was unfortunately not fully met. In Figure A-2, it is possible to see the final *Gantt Chart* containing the actual time required to finish this project.

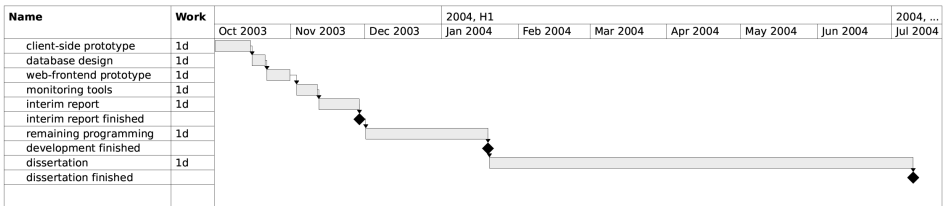


Figure A-2. Final Gantt Chart

The only real difference between the time planned and that actually required to finish this project is the time required to finish this dissertation. The design of the project, the programming and testing of the different parts was more or less finished in time. The writing of this paper, however, was the part which required much more time than expected. There are multiple reasons why it required more time than estimated. The first reason is that I underestimated the amount of time required to write the dissertation. It was not so much the actual writing of the dissertation than, for example, creating the structure of the dissertation or thinking of the possible alternatives I did not choose. The second reason for the delay was the lack of motivation which led to periods of sometimes up to three weeks when nothing happened. The third reason for this large delay is that I just did not have the time to work on the dissertation as it had to be done in my spare time and sometimes there was just not enough spare time left. A big error on my side was that I seriously underestimated the time needed to write the dissertation. But finally I was able to finish the dissertation, meeting the aims (regarding content and time) I set for myself regarding this paper as well as the programming.

Another small difference between the original time plan and the actual time plan is that task *remaining programming* required two weeks more than expected. This was the result from the first task *client-side prototype* which required to be rewritten to

meet the new aims which have changed during the dissertation due to results during the development.

A.2. Project Management

This chapter is not the actual project management but about all the programmes and tools used during this project, programmes for writing the code, for writing the dissertation, programmes for supporting a version control of code and the dissertation etc.

As I keep mentioning how great Free Software is, all the parts of this dissertation are of course are only created and managed using Free Software. The following is a list of many (but surely not all) Free Software programmes used during this dissertation. It is meant as a recognition of these Free Software projects and thanks to the many people working on it:

- *Linux*, of course, as the operating system, in particular Debian (http://www.debian.org/) and Fedora (http://fedora.redhat.com/)
- *Vim* as my editor of choice to write all the code and this dissertation. (http://www.vim.org/)
- *CVS* is the Concurrent Versions System. *CVS* is used for the version control of all the code and this dissertation. (http://www.cvshome.org/)
- *PHP* has been already mentioned. It is the programming language used to create the web-interface. (http://www.php.net/)
- *Perl* has also already been mentioned. This is the programming language used for the mirror monitoring tool. (http://www.perl.org/)
- *Python* is the programming language for the GUI used in combination with

PyGTK. (http://www.python.org/ and http://www.pygtk.org/)

- *Apache HTTP Server Project* is used as the web-server required by many parts of the dissertation. (http://httpd.apache.org/)
- *PostgreSQL* is used as the database for storing the relevant data. (http://www.postgresql.org)
- *DocBook* is used to create this document. (http://www.docbook.org)
- *Dia* is used to create all the diagrams in this dissertation. (http://www.gnome.org/projects/dia/)
- And thanks to all the other tools I have used to create this dissertation

A.3. See It For Yourself

The following is a list of where the software and the source-code created during this dissertation can be seen:

- *http://lisas.de/~adrian/master/* - this is the URL where the web-front-end can be seen in action.
- *http://lisas.de/~adrian/master/code/* - this is the URL showing the source-code created during this dissertation.
- *http://lisas.de/~adrian/master/dissertation.pdf* - this document.

Appendix B. Source Code

B.1. checkmirrors.pl

```
#!/usr/bin/perl -w
#
# checkmirrors.pl - check the mirrors
# Copyright (C) 2004 Adrian Reber
# $Id: checkmirrors.pl,v 1.4 2004/09/30 15:20:58 adrian Exp $

# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2, or (at your option)
# any later version.

# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.

# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA
# 02111-1307, USA.

use strict;
use DBI;
use LWP;
use HTTP::Request;
use HTTP::Status;

my $dbname = “adrian”;

my $outfile        = “/home/staff/adrian/.public_html/fedora-mirrors.list”;
my $static_mirrors = “/home/staff/adrian/mirror_list.static”;
my $zonefile       = “/home/staff/adrian/.public_html/db.mirrors”;

# the number of days after which entries from the database should be removed
my $delete_if_older = 7;

my $ua = LWP::UserAgent->new( timeout => 30 );
my $request;
my $response;
my @row;
my $sth;
my %protocols;
```

```
my %mirror_hosts;
my %mirror_description;
my %mirror_host;
my %mirror_path;
my %mirror_protocol;
my %mirror_country;
my %dns_hash;
my %dns_country_hash;
my $master_timestamp = -1;
my $timestamp        = -1;
my $older            = -1;
my $debug            = 0;
my $key;
my $number_of_probes;
my $old_age;
my $kill_age;

if ( $#ARGV == 0 ) {
    $debug = 1;
}

my $dbh = DBI->connect( “dbi:Pg:dbname=$dbname”, “”, “” );

sub write_mirror_list_static {
    open( STATIC_MIRRORS, “<$static_mirrors” )
      || die “can’t open $static_mirrors for reading\n$!\n”;
    open( OUTFILE, “>$outfile” ) || die “can’t open $outfile for writing\n$!\n”;
    while (<STATIC_MIRRORS>) {
        chomp;
        print OUTFILE $_, “\n”;
    }
    print OUTFILE “Fedora Extras”;

    close(OUTFILE);
    close(STATIC_MIRRORS);
}

sub write_mirror_list {
    my $age         = $_[0];
    my $is_old      = $_[1];
    my $description = $_[2];
    my $protocol    = $_[3];
    my $host        = $_[4];
    my $path        = $_[5];
    open( OUTFILE, “>>$outfile” )
      || die “can’t open $outfile for writing\n$!\n”;

    print OUTFILE “\n”;
    if ( length($description) > 0 ) {
        print OUTFILE $description;
    }
    else {
        print OUTFILE $host;
```

```
    }
    print OUTFILE " (", $protocol, ")";
    if ( $is_old == 1 ) {
        printf OUTFILE " (mirror not updated in the last %2.1f hours)",
          $age / 3600;
    }
    print OUTFILE "\n", $protocol, "://", $host, $path,
      "fedora fedora/1/i386 stable\n";
    print OUTFILE "--";
    close(OUTFILE);
}

sub start_write_zone_file {

    my $localtime = localtime();
    open( OUTFILE, ">$zonefile" )
      || die "can't open $outfile for writing\n$!\n";
    print OUTFILE "; autogenerated by checkmirrors.pl\n";
    print OUTFILE "; ", $localtime, "\n\n";

    close(OUTFILE);
}

sub write_zone_file {
    my $description = $_[0];
    my $host        = $_[1];
    my $country     = $_[2];
    if ( !defined $description ) {
        $description = "";
    }
    if ( !defined $country ) {
        $country = "";
    }
    $dns_hash{$host}         = $description;
    $dns_country_hash{$host} = $country;
}

sub end_mirror_list {
    open( OUTFILE, ">>$outfile" )
      || die "can't open $outfile for writing\n$!\n";

    print OUTFILE "--\n";
    close(OUTFILE);
}

sub end_zone_file {

    open( OUTFILE, ">>$zonefile" )
      || die "can't open $outfile for writing\n$!\n";
    foreach $key ( keys %dns_hash ) {
        ( my $name, my $aliases, my $addrtype, my $length, my @addrs ) =
          gethostbyname($key);
        ( my $a, my $b, my $c, my $d ) = unpack( 'C4', $addrs[0] );
```

```
        my $ip = “$a.$b.$c.$d”;
        print OUTFILE “ftp”;
        if ( length( $dns_country_hash{$key} ) > 0 ) {
            print OUTFILE “.”, $dns_country_hash{$key};
        }
        print OUTFILE “\t\t\tA\t”, $ip, “\n”;
        print OUTFILE “\t\t\tTXT\t\””;
        if ( length( $dns_hash{$key} ) > 0 ) {
            print OUTFILE $dns_hash{$key};
        }
        else {
            print OUTFILE $key;
        }
        print OUTFILE ”\“\n”;
    }

    close(OUTFILE);
}

sub end_write {
    end_mirror_list;
    end_zone_file;
}

sub start_write {
    write_mirror_list_static();
    start_write_zone_file();
}

sub write_generic {
    my $age         = $_[0];
    my $is_old      = $_[1];
    my $is_dead     = $_[2];
    my $description = $_[3];
    my $protocol    = $_[4];
    my $host        = $_[5];
    my $path        = $_[6];
    my $country     = $_[7];
    if ( $debug == 1 ) {
        print “write_generic, $age, $is_old, $is_dead\n”;
    }
    if ( $is_dead == 0 ) {
        write_mirror_list( $age, $is_old, $description, $protocol, $host,
            $path );
        write_zone_file( $description, $host, $country );
    }
}

sub read_settings() {
    my $sth = $dbh->prepare(“select name, value from settings”);
    $sth->execute();
    while ( @row = $sth->fetchrow_array ) {
        if ( $row[0] =~ /Number_of_probes/ ) {
```

```
            $number_of_probes = $row[1];
        }
        elsif ( $row[0] =~ /Old_Age/ ) {
            $old_age = $row[1];
        }
        elsif ( $row[0] =~ /Kill_Age/ ) {
            $kill_age = $row[1];
        }
    }
}

sub generate_mirror_list() {
    start_write();
    my $sth =
      $dbh->prepare( “select tstamp,age from monitoring ”
          . “ where mirror_id=? order by tstamp desc limit $number_of_probes” );
    foreach $key (
        sort { $mirror_hosts{$b} <=> $mirror_hosts{$a} }
        keys %mirror_hosts
      )
    {
        $sth->execute($key);
        my $first       = 0;
        my $rather_old  = 0;
        my $too_old     = 0;
        my $current_age = -1;
        while ( @row = $sth->fetchrow_array ) {
            if ( $first == 0 || $current_age < 0 ) {
                $first       = 1;
                $current_age = $row[1];
            }
            if ( $row[1] >= $old_age ) {
                $rather_old++;
            }
            if ( $row[1] >= $kill_age || $row[1] < 0 ) {
                $too_old++;
            }
        }
        if ( $too_old >= $number_of_probes ) {
            $too_old = 1;
        }
        else {
            $too_old = 0;
        }
        if ( $rather_old >= $number_of_probes ) {
            $rather_old = 1;
        }
        else {
            $rather_old = 0;
        }

        write_generic(
            $current_age,           $rather_old,
```

```
            $too_old,               $mirror_description{$key},
            $mirror_protocol{$key}, $mirror_host{$key},
            $mirror_path{$key},     $mirror_country{$key}
        );
    }
    end_write();
}

sub get_master_timestamp() {
    my $sth =
      $dbh->prepare("SELECT value from settings where name='masterurl'");
    $sth->execute();
    my @row = $sth->fetchrow_array;
    $request = HTTP::Request->new( GET => "$row[0]TIMESTAMP" );
    if ($debug) {
        print "sending request to: ", $request->uri(), "\n";
    }
    $response = $ua->request($request);
    if ($debug) {
        print "sending request done with: ", $response->code(), "\n";
    }
    if ( $response->code() == RC_OK ) {
        $master_timestamp = $response->content();
        chomp $master_timestamp;
    }
}

sub read_protocols() {
    my $sth = $dbh->prepare("SELECT protocol_id, protocol from protocol");
    $sth->execute();
    my @row;
    while ( @row = $sth->fetchrow_array ) {
        $protocols{ $row[0] } = $row[1];
    }
}

sub cleanup_db() {
    my $age = $delete_if_older * 86400;
    my $now = time();
    $age = $now - $age;
    $sth = $dbh->prepare("DELETE from monitoring where tstamp < $age");
    $sth->execute();
}

$sth =
  $dbh->prepare( "SELECT p.protocol, m.host, m.path, m.mirror_id, "
      . " m.description, m.country from mirrors m, "
      . " protocol p where p.protocol_id=m.protocol" );

$sth->execute();

read_settings();
```

```
get_master_timestamp();

read_protocols();

if ( $master_timestamp == -1 ) {
    print “no timestamp from the master mirror server.\n”;
    die “exiting”;
}

while ( @row = $sth->fetchrow_array ) {
    $request = HTTP::Request->new( GET => “$row[0]://$row[1]$row[2]TIMESTAMP” );
    if ($debug) {
        print “sending request to: ”, $request->uri(), “\n”;
    }
    $response = $ua->request($request);
    if ($debug) {
        print “sending request done with: ”, $response->code(), “\n”;
        print “description: ”, $row[4], “\n”;
    }
    $mirror_description{ $row[3] } = $row[4];
    $mirror_protocol{ $row[3] }    = $row[0];
    $mirror_host{ $row[3] }        = $row[1];
    $mirror_path{ $row[3] }        = $row[2];
    $mirror_country{ $row[3] }     = $row[5];
    if ( $response->code() == RC_OK ) {
        $timestamp = $response->content();
        chomp $timestamp;
        $older = $master_timestamp - $timestamp;
        $mirror_hosts{ $row[3] } = $older;
    }
    else {
        $mirror_hosts{ $row[3] } = -$response->code();
    }
}

$sth =
  $dbh->prepare(
“INSERT INTO MONITORING(MIRROR_ID,TSTAMP,AGE,MASTER_TIMESTAMP) VALUES (?,?,?,?)”
  );

my $now = time();

foreach $key (
    sort { $mirror_hosts{$b} <=> $mirror_hosts{$a} }
    keys %mirror_hosts
  )
{
    $sth->execute( $key, $now, $mirror_hosts{$key}, $master_timestamp );
}

generate_mirror_list();
cleanup_db();
$dbh->disconnect;
```

B.2. mirror-setup.py

```
#!/usr/bin/python

# $Id: mirror-setup.py,v 1.1 2004/09/30 15:26:12 adrian Exp $
#
# select a mirror
# Copyright (C) 2004 Adrian Reber

# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2, or (at your option)
# any later version.

# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.

# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA
# 02111-1307, USA.

import sys
import os
import commands
import threading
import urllib
import gobject
import gtk
import getopt
import urllib
import string
import time
from gtk import *

version = “0.1”
name = “mirror-setup”
result = “NORESULT”
sema4 = threading.Semaphore()
fn= “”
netselect=“/usr/bin/netselect”
mirrorlist = “http://lisas.de/~adrian/fedora-mirrors.list”
```

```
display_mirrors = “3”

class SourcesListWriter:

        def __init__(self, fn=None):
      if fn is None:
         self.fn = “sources.list”
      else:
         self.fn = fn
      self.list = {}
      self.priority = {}

   def add(self,key,item,priority):
      self.list[key] = item
      prio_str = “%d” % priority
      self.priority[key] = prio_str

   def remove(self,key):
      del self.list[key]
      del self.priority[key]

   def write(self):
      fp=open(self.fn,“w”)
      currtime = time.strftime(“%x %X %Z”, time.localtime())
      fp.write(“### generated from %s ###\n” % name)
      fp.write(“### %(currtime)s\n\n” % vars())
      #find lowest priority
      priority = {}
      for key in self.priority.keys():
         temp = string.atoi(self.priority[key])
         priority[temp] = key
      thekeys = priority.keys()
      thekeys.sort()
      for key in thekeys:
         real_key = priority[key]
         fp.write(“### ” +  self.list[real_key] + “\n”)
         fp.write(“rpm ” + real_key + “\n\n”)
      fp.close()

class Mirror:

   def __init__(self):
      self.url = ”
      self.comment = ”

   def setUrl(self,url):
      self.url = url

   def setComment(self,comment):
      self.comment = comment

   def getUrl(self):
      return self.url
```

```
   def getComment(self):
      return self.comment


class Mirrors:

   def __init__(self,data):
      self.mirrors = list()
      self.parse(data)

   def parse(self,data):
      found = 0
      comment = ”
      url = ”
      for line in data:
         if (line==“Fedora Extras”):
            found = 1
            continue
         if (found==1 and comment==”):
            comment = line
            continue
         if (found==1 and url==”):
            url = line
            continue
         if (found==1):
            self.addMirror(comment,url)
            comment=”
            url=”



   def addMirror(self,comment,url):
      mirror = Mirror()
      mirror.setUrl(url)
      mirror.setComment(comment)
      self.mirrors.append(mirror)

   def getSize(self):
      return len(self.mirrors)

   def getUrl(self,index):
      return self.mirrors[index].getUrl()

   def getComment(self,index):
      return self.mirrors[index].getComment()

class NetSelect(threading.Thread):

   def __init__(self, hosts):
      threading.Thread.__init__(self)
      self.hosts = hosts
```

```
   def run(self):
      sema4.acquire()
      global result
      if (os.path.isfile(netselect)):
         command = netselect + " -s " + display_mirrors + " "
         for key in self.hosts.keys():
            command += key + " "
         output = commands.getoutput(command)
         result = output
      else:
         result = netselect + " not found!"
      sema4.release()

class MirrorSelector:

   counter = 0
   netselect_text = "Searching for fastest mirror\n"

   def check_thread(self):
      if (self.counter >=7):
         self.counter = 0
      global result
      self.message.label.set_text(self.netselect_text + "Please wait" + ". " * self.counter)
      self.counter = self.counter+1
      if (sema4.acquire(FALSE)==FALSE):
         return TRUE
      sema4.release()
      self.counter = 0
      localresult = string.split(result,"\n")
      self.ok_button.set_sensitive(TRUE)
      label = self.netselect_text + "Displaying the %d fastest mirrors:\n" % len(localresult)
      i = 0
      for line in localresult:
         button = gtk.CheckButton()
         try:
            button = self.checkbuttons[self.parameter[line[6:]]]
            i = i + 1
            self.priority[line[6:]] = i
            button.set_active(gtk.TRUE)
            label += line[6:] + "\n"
         except KeyError:
            label = self.netselect_text + result
      self.message.label.set_text(label)
      result = "NORESULT"
      return FALSE

   def netselect(self, widget, data=None):
      for key in self.priority.keys():
         self.priority[key] = 0
      if (sema4.acquire(FALSE)==FALSE):
         return
      sema4.release()
      net_select = NetSelect(self.parameter)
```

```
   net_select.start()
   gobject.timeout_add(200,self.check_thread)
   self.message.label.set_text(self.netselect_text)
   self.ok_button.set_sensitive(FALSE)
   self.message.run()
   self.message.hide()

def write(self, widget, data=None):
   self.writer.write()

def checked(self, widget, event, data):
   if (widget.get_active()):
      url = string.split(self.mirrors.getUrl(data),'/')
      self.writer.add(self.mirrors.getUrl(data),self.mirrors.getComment(data),
         self.priority[url[2]])
   else:
      self.writer.remove(self.mirrors.getUrl(data))

def delete_event(self, widget, event, data=None):
   return gtk.FALSE

def destroy(self, widget, data=None):
   gtk.main_quit()

def create_table(self):
   scrolled_window = gtk.ScrolledWindow()
   scrolled_window.set_policy(gtk.POLICY_AUTOMATIC, gtk.POLICY_AUTOMATIC)
   self.vbox.add(scrolled_window)
   table = gtk.Table(self.mirrors.getSize(),2,gtk.FALSE)
   label = gtk.Label("Fedora Mirror")
   table.attach(label, 0, 1, 0, 1)
   label = gtk.Label("choose")
   table.attach(label, 1, 2, 0, 1,0,0,5,5)
   for x in range(self.mirrors.getSize()):
      text = string.replace(mirrors.getComment(x),'(mirror','\n(mirror')
      url = string.split(mirrors.getUrl(x),'/')
      self.parameter[url[2]] = x
      self.priority[url[2]] = 0
      label = gtk.Label(text)
      label.set_justify(gtk.JUSTIFY_CENTER)
      table.attach(label, 0, 1, x+1, x+2, 0, 0,3,3)
      button = gtk.CheckButton()
      button.set_active(gtk.FALSE)
      table.attach(button,1,2,x+1, x+2, 0 ,0,3,3)
      button.connect("clicked",self.checked, None,x)
      self.checkbuttons[x] = button
   scrolled_window.add_with_viewport(table)
   scrolled_window.set_size_request(500,300)

def add_buttons(self):
   buttonbox = gtk.HButtonBox()
   self.vbox.add(buttonbox)
   self.measure_tip = gtk.Tooltips()
```

```
        measure = gtk.Button(“Measure”)
        write = gtk.Button(“Write”)
        exit = gtk.Button(“Exit”)
        exit.connect_object(“clicked”, gtk.Widget.destroy, self.window)
        write.connect(“clicked”,self.write, None)
        measure.connect(“clicked”,self.netselect, None)
        buttonbox.pack_start(measure)
        buttonbox.pack_start(write)
        buttonbox.pack_start(exit)
        self.measure_tip.set_tip(measure,“Measure which mirror is the closest”)
        self.measure_tip.set_tip(exit,“Quit from %s” % name)
        buttonbox.set_layout(gtk.BUTTONBOX_SPREAD)


    def __init__(self, mirrors):
        self.checkbuttons = {}
        self.parameter = {}
        self.priority = {}
        self.mirrors = mirrors
        self.window = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.window.connect(“delete_event”, self.delete_event)
        self.window.connect(“destroy”, self.destroy)
        self.message = gtk.MessageDialog(self.window,gtk.DIALOG_DESTROY_WITH_PARENT,gtk.MESSAGE_INFO,gtk.BUTTONS_NONE)
        self.ok_button = self.message.add_button(gtk.STOCK_OK, gtk.RESPONSE_OK)
        self.vbox = gtk.VBox()
        self.vbox.set_homogeneous(FALSE)
        self.window.add(self.vbox)
        self.create_table()
        self.add_buttons()
        self.window.show_all()
        self.writer = SourcesListWriter(fn)
        self.window.set_title(name)

    def main(self):
        gtk.gdk.threads_init()
        gtk.main()

userknowsitbetter=0

def weknowitbetter():
    if (userknowsitbetter==0):
        print “This program should be run as root.”
        print “But if you think you know what you are doing”
        print “you can specify ’--iknowitbetter’ on the command line.”
        sys.exit(1)
    else:
        print “So you think you know it better...”
        print “Okay, let’s see if you really do”
        print “Running as non root user...”

def usage():
    print >> sys.stderr, “””
Usage: %s [options] [file]
```

```
Options:
  -h, --help          print this message
  -m, --mirrorlist    URL of the mirrorr list
                      defaults to:
          %s
  -n, --number        specify the number of mirrors used
                      defaults to: %d

Argumemt:
  file                file for writing output
                      default: ./sources.list
           file will be overwritten
""" % (name, mirrorlist, display_mirrors)

print >> sys.stderr, """%s Release %s - Copyright (C) 2004 Adrian Reber
%s comes with ABSOLUTELY NO WARRANTY - for details read the license.
""" % (name, version, name)


try:
   o, a = getopt.getopt(sys.argv[1:], 'm:hn:',['iknowitbetter','help','mirrorlist=','number='])
   opts = {}
except:
   usage()
   sys.exit(0)

for k,v in o:
   opts[k] = v
if (opts.has_key("-h") or opts.has_key("--help")):
   usage()
   sys.exit(0)
if (opts.has_key("-n") or opts.has_key("--number")):
   display_mirrors=opts[k]
if (opts.has_key("-m") or opts.has_key("--mirrorlist")):
   mirrorlist=opts[k]

if opts.has_key('--iknowitbetter'):
   userknowsitbetter=1

if len(a) < 1:
   fn = "sources.list"
else:
   fn=a[0]

if (os.getuid()!=0):
   weknowitbetter()

try:
   f = urllib.urlopen(mirrorlist)
except IOError, (errno, strerror):
   print "I/O error(%s): %s\n" % (errno, strerror)
   print "Maybe you need to specify a proxy:"
```

```
   print “export http_proxy=\”http://proxy:port/\“\n”
   sys.exit(1)

data = string.split(f.read(), ’\n’)

mirrors = Mirrors(data)

mirrorSelector = MirrorSelector(mirrors)
try:
   mirrorSelector.main()
except KeyboardInterrupt:
   print “Aborted!”
```

B.3. Web Interface (functions.inc)

As the source code concerning the web interface is much too long to be presented here only one file is printed representing the others.

```
<?php
   # $Id: functions.inc,v 1.2 2004/09/30 16:20:05 adrian Exp $
        function dbconnect() {
                $dbconn = pg_connect(“host=localhost dbname=mirman user=user password=password”);
                if (!$dbconn) {
                        echo “Connection to database could not be established.\n”;
                        return null;
                }
                return $dbconn;
        }

        function htmlfooter() {
      echo “<p class=\”date\“>Page creation time: ”;
      echo gmdate(’D, d M Y H:i:s \G\M\T’, time());
      echo “</p>\n”;
      echo “<hr/><p class=\”fortune\“>\n”;
      echo passthru(“/usr/bin/fortune calvin”);
      echo “</p>”;
                echo “</body>\n</html>”;
        }

   function make_nice_url($link) {
              $link = ereg_replace(“://”,“xxx/xxx”,$link);
              $link = ereg_replace(“//”,“/”,$link);
              $link = ereg_replace(“//”,“/”,$link);
              $link = ereg_replace(“//”,“/”,$link);
              return ereg_replace(“xxx/xxx”,“://”,$link);
```

```
}

     function not_logged_in() {
   $rootdir = “”;
   if ($_SESSION[“rootdir”])
      $rootdir = $_SESSION[“rootdir”];
             if ($_SESSION[“uid”] == “”) {
                     echo “You are not logged in.<br>”;
                     echo “Login at the <a href=\””. $rootdir .”\”>login page</a>”;
                     htmlfooter();
                     return true;
             }
             return false;
     }

function is_admin() {
   if ($_SESSION[“admin”] == “0”) {
      echo “You are not an admin.<br>”;
      htmlfooter();
      return true;
   }
   else
      return false;
}

     function authenticated($uid, $password) {
             global $dbconn;
             if ($password == “” || $uid == “”)
                     return false;
             $result = pg_query($dbconn, “select * from users”);
             if (!$result) {
                     echo “Query failed.\n”;
                     pg_close($dbconn);
                     return false;
             }

             while ($arr = pg_fetch_array($result)) {
                     if ($arr[“uid”] == $uid && $arr[“password”] == md5($password)) {
                             $_SESSION[“uid”] = $arr[“uid”];
                             $_SESSION[“firstname”] = $arr[“firstname”];
                             $_SESSION[“lastname”] = $arr[“lastname”];
                             $_SESSION[“email”] = $arr[“email”];
                             $_SESSION[“admin”] = $arr[“admin”];
                             return true;
                     }
             }
             return false;
     }

function password($length = “8”){
   $password = NULL;
   $tmp = NULL;
   for($i=0; $i<$length; $i++) {
```

```
          $character = chr(rand(48,122));
      while (!ereg("[a-zA-Z0-9]", $character)){
         if($character == $tmp) continue;
         $character = chr(rand(48,90));
      }
      $password .= $character;
      $tmp = $character;
   }
   return $password;
}

function sendmail($to, $subject, $message) {
            global $dbconn;
   $select = "select * from settings";
   $result = pg_query($dbconn, $select);

   while ($arr = pg_fetch_array($result)) {
      if ($arr["name"] == "admin_email")
         $admin_email = $arr["value"];
      if ($arr["name"] == "rootdir")
         $rootdir = $arr["value"];
   }

   $arr = posix_uname();
   $header = "Date: " . date("D, j M Y G:i:s") . "\n";
   $header .= "From: Mirror Management <" . $admin_email  . ">\n";
   $header .= "User-Agent: Mirror Management/0.1\n";
   $header .= "X-Url: http://" . $_SERVER["SERVER_NAME"] . $rootdir . "\n";
   $header .= "X-Operating-System: " . $arr["sysname"] . " (" . $arr["release"] . ")\n";
   $header .= "X-Unexpected: The Spanish Inquisition\n";
   $header .= "Message-Id: <" . time() . "@" . $_SERVER["SERVER_NAME"] . ">\n";
   mail($to, $subject, $message, $header);
}

function linkbar() {
   $rootdir = "";
   if ($_SESSION["rootdir"])
      $rootdir = $_SESSION["rootdir"];
   echo '<table width="100%"><tr><td> </td><td align="right">';
   echo '|<a href="' . $rootdir . '">Status</a>';
   echo '|<a href="' . $rootdir . '/releases.php">Releases</a>';
   if ($_SESSION["login"] != 0) {
      echo '|<a href="' . $rootdir . '/mymirrors.php">My Mirrors</a>';
      echo '|<a href="' . $rootdir . '/password.php">Change Password</a>';
      echo '|<a href="' . $rootdir . '/change.php">My Account</a>';
      if ($_SESSION["admin"] == "1") {
         echo '|<a href="' . $rootdir . '/admin/">Admin Home</a>';
      }
      echo '|<a href="' . $rootdir . '/logout.php">Logout</a>';
   } else {
      echo '|<a href="' . $rootdir . '/login.php">Login</a>';
   }
   echo '|</td></tr></table>';
```

```
   }

   function admin_navigation() {
      echo '<table><tr><th nowrap>Admin Navigation</th></tr>';
      echo '<tr><td nowrap><a href="users.php">User Administration</a></td></tr>';
      echo '<tr><td nowrap><a href="mirrors.php">Mirror Administration</a></td></tr>';
      echo '<tr><td nowrap><a href="settings.php">Global Settings</a></td></tr>';
      echo '<tr><td nowrap><a href="releases.php">Release Administration</a></td></tr>';
      echo '</table>';
   }

        function htmlheader() {
        ?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
        <head>
                <title>Mirror</title>
                <link rel="stylesheet" href="main.css" type="text/css">
                <meta http-equiv="content-type" content="text/html; charset=iso-8859-1">
      <link rel="shortcut icon" href="favicon.png" TYPE="image/png">
        </head>
<body>
        <?php
        }
?>
```

B.4. checkmirrorpermission.pl

```
#!/usr/bin/perl -w
#
# checkmirrors.pl - check the mirrors
# Copyright (C) 2004 Adrian Reber
# $Id: checkmirrorpermission.pl,v 1.2 2004/09/30 16:08:47 adrian Exp $

# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2, or (at your option)
# any later version.

# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.

# You should have received a copy of the GNU General Public License
```

```
# along with this program; if not, write to the Free Software
# Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA
# 02111-1307, USA.

use strict;
use DBI;
use LWP;
use HTTP::Request;
use HTTP::Status;

my $dbname = “mirman”;

my $ua = LWP::UserAgent->new(
    timeout => 30,
    agent   => “Another mirror checker (http://lisas.de/mirman/)”
);
my $request;
my $response;
my @row;
my $sth;
my $path;
my %release_paths;
my %release_archs;
my %readable;
my %protocols;
my %mirror_hosts;
my %mirror_description;
my %mirror_host;
my %mirror_path;
my %mirror_protocol;
my %mirror_country;
my %dns_hash;
my %dns_country_hash;
my $master_timestamp = -1;
my $timestamp        = -1;
my $older            = -1;
my $debug            = 0;
my $permission_check;
my $key;
my $number_of_probes;
my $old_age;
my $kill_age;

if ( $#ARGV == 0 ) {
    $debug = 1;
}

my $dbh = DBI->connect( “dbi:Pg:dbname=$dbname”, “”, “” );

sub read_settings() {
    my $sth = $dbh->prepare(“select name, value from settings”);
    $sth->execute();
    while ( @row = $sth->fetchrow_array ) {
```

```
        if ( $row[0] =~ /Number_of_probes/ ) {
            $number_of_probes = $row[1];
        }
        elsif ( $row[0] =~ /Old_Age/ ) {
            $old_age = $row[1];
        }
        elsif ( $row[0] =~ /Kill_Age/ ) {
            $kill_age = $row[1];
        }
        elsif ( $row[0] =~ /permission_check/ ) {
            $permission_check = $row[1];
        }
    }
}

sub read_protocols() {
    my $sth = $dbh->prepare("SELECT protocol_id, protocol from protocol");
    $sth->execute();
    my @row;
    while ( @row = $sth->fetchrow_array ) {
        $protocols{ $row[0] } = $row[1];
    }
}

sub get_releases() {
    my $sth = $dbh->prepare("SELECT release_id,path,arch from releases");
    $sth->execute();
    my @row;
    my $i = 0;
    while ( @row = $sth->fetchrow_array ) {
        $release_paths{ $row[0] } = $row[1];
        $release_archs{ $row[0] } = $row[2];
    }
}

$sth =
  $dbh->prepare(
        "SELECT p.protocol, m.host, m.path, m.mirror_id, m.description, "
      . " m.country, rs.release_id, rd.released_id from mirrors m, protocol p, "
      . " releases rs, released rd where p.protocol_id=m.protocol and rs.check_readable=1 "
      . " and rd.mirror_id=m.mirror_id and rs.release_id = rd.release_id" );

$sth->execute();

read_settings();

read_protocols();

get_releases();

while ( @row = $sth->fetchrow_array ) {
    $path =
"$row[1]$row[2]/$release_paths{$row[6]}/$release_archs{$row[6]}/$permission_check/";
```

```
    $path =~ s/\/+/\//go;
    $request = HTTP::Request->new( GET => “$row[0]://$path” );
    $request =~ s/\/+/\//go;
    if ($debug) {
        print “sending request to: ”, $request->uri(), “\n”;
    }
    $response = $ua->request($request);
    if ($debug) {
        print “code: ”, $response->code(), “\n”;
    }
    if ( $response->code() == RC_OK ) {
        $readable{ $row[7] } = RC_OK;
    }
    next;

    if ($debug) {
        print “sending request done with: ”, $response->code(), “\n”;
        print “description: ”, $row[4], “\n”;
    }
    $mirror_description{ $row[3] } = $row[4];
    $mirror_protocol{ $row[3] }    = $row[0];
    $mirror_host{ $row[3] }        = $row[1];
    $mirror_path{ $row[3] }        = $row[2];
    $mirror_country{ $row[3] }     = $row[5];
    if ( $response->code() == RC_OK ) {
        $timestamp = $response->content();
        chomp $timestamp;
        $older = $master_timestamp - $timestamp;
        $mirror_hosts{ $row[3] } = $older;
    }
    else {
        $mirror_hosts{ $row[3] } = -$response->code();
    }
}

$sth = $dbh->prepare( “UPDATE RELEASED SET READABLE=1 WHERE RELEASED_ID=?” );

my $now = time();

foreach $key ( keys %readable ) {
    $sth->execute($key);
}

$dbh->disconnect;
```

Interim Report

Distributed Software Installation for Free Software

Interim Report for the MSc Dissertation in Distributed Systems

Adrian Reber

$Id: interim.src.xml,v 1.3 2003/12/08 20:48:31 adrian Exp $

Table of Contents

1. Introduction 2
2. Background 4
3. Initial Survey 7
4. Aims and Objectives 9
5. Initial Design 9
    5.1. Database 10
    5.2. Monitoring 12
    5.3. Client Side 14
6. Time Plan 15
7. Outcome 15
References 15
Bibliography 16

1. Introduction

This is the interim report for the MSc Dissertation in Distributed Computing Systems Engineering on the topic: *Distributed Software Installation for Free Software*

Most currently available Linux Distributions which have a large user-base are backed by a commercially operating company. The one big exception to this pattern is Debian[1], which is run entirely by the Linux community. The great

advantage of such a community-based model is that it is very easy for new software to be added to this Linux Distribution as a lot of people are involved and it is not hard to find a maintainer for the new software package.

As an example of a commercially based Linux Distribution, Red Hat[2] is used. Red Hat is one of the few Linux Distributions, if not the only one, which works independently and makes a profit using Free Software[3].

For a distribution like Red Hat, it is not particularly easy to include software in its distribution as it lives from supporting the Red Hat Linux Distribution and a new software package means an additional part which needs to be supported and therefore requires additional manpower involving additional expenses.

As the community-based approach works very well and efficiently for Debian, some people started something similar, but based on the Red Hat Linux Distribution. As all the parts of both Linux Distributions are based on Free Software, the usual approach for the software installation is to download it from any of the available mirror servers[1]. As Debian is already 10 years old, the infrastructure for software distribution and automatic installation already exists. For the community-based parts of Red Hat, however, such a complete infrastructure does not yet exist and this is the starting point of this dissertation. Of course, there are already some ideas on how to handle the software distribution. At the present there is still a lot of development to be done, so this is a good time to offer a complete solution which is to be developed in this dissertation. Most of

[1]A mirror server is system which mirrors the data from one server to increase the availability and accessibility of the data; the data on a mirror server are usually not altered and depend on the mirrored data being updated regularly.

the current mechanisms are based upon ideas from Debian or from experience with processes which did not work as expected and therefore do not need to be tried again or, if so, only in an improved form.

2. Background

One of the big problems in software distribution, and this is not only the case with Free Software, is the dependence of one software package on another. The most common case is that one program requires multiple shared libraries which are linked dynamically during runtime. Usually, the package manager will check the local database containing the installed software to find out whether all the software required for the current package is already provided by the system; if this check is positive, the software will be installed. If the check is negative, the software will not be installed and a message will inform the user which requirements for the software installation have not been fulfilled by the system.

The solution to this problem is that the package manager on the local system has to have a database of all packages available on the mirror servers. In this way, the package manager can resolve all the dependencies by downloading all the necessary software packages and install them in the best order. As this is not really the right task for the package manager, the most logical solution is to write a front-end for the package manager which uses the package manager to install new software but has its own database which it uses to resolve all the dependencies. This is exactly what has been done with Debian.

Debian uses *dpkg* as its package manager and *apt*[2] is the front-end which

handles all the dependency resolutions, ordering the package and, if necessary, downloading the package from any of the Debian mirror servers. The tool *apt* works very well and is one of the reasons Debian is so popular. It is therefore not very surprising that the users of Red Hat demanded something similar. As *apt* is Free Software and the source code is of course available, it is very obvious that *apt* was changed so as to work on Red Hat-based systems. This is exactly what Conectiva[4] did. Conectiva is also a Linux Distribution which is based on the package manager from Red Hat: *rpm*[3].

It was not long after Conectiva released their port of *apt* for *rpm*-based Linux Distributions that people started to use *apt* not only with Conectiva Linux but with many other *rpm*-based Linux Distributions. The first step in using *apt* with *rpm* was to make some of the mirror servers for Red Hat *apt*-aware. This enabled everybody who wanted to test *apt* to install any package available in the distribution. Now that every package available in the distribution was only one command away the demand for bigger software repositories, just like Debian, started to arise. As nobody expected Red Hat to create such software repositories, some people started creating software repositories on their own. The following is a list of the larger and better-known repositories:

- http://freshrpms.net/
- http://atrpms.physik.fu-berlin.de/
- http://newrpms.sunsite.dk/

[2]Advanced Package Tool

[3]rpm - *R*ed Hat *P*ackage *M*anager

- http://apt.sw.be/

Repositories like these were the first step in providing a bigger number of software packages. Although these repositories offer a good selection of high-quality packages, there were still some problems. In the beginning, all these repositories were developed independently and a lot packages were maintained in multiple repositories. The repositories became incompatible due to different naming schemes of the packages and similar packaging-related problems. As a result a user either had to change the configuration which repositories to use often or just opt for one of them and not use the others. Another problem was that each of these repositories was maintained by another person who had personal preferences and usually only packaged what they needed. Of course, the maintainers responded to package requests but some people were unhappy that there was not any really well defined process on how to add a package to a repository. This is also one of the big advantages of Debian. Each package is usually only maintained by the person who is interested in it and therefore most of the packages are very well maintained.

That was the point at which a group of people decided that, if it was possible for Debian to maintain a extremely large pool of packages with well defined processes for package creation, bug reports and quality assurance, this must of course also be feasible for a software repository based on Red Hat. This was the birth of the Fedora Project [5].

The goal of the Fedora Project is to create high-quality third-party add-on packages for the Red Hat Linux Distribution. To achieve this goal, many discussions were necessary to decide on many of details. Maybe there were too many dis-

cussions, but the goal was to define rules and processes for every detail of the Fedora Project, so that it would not be necessary to learn from mistakes. After some months, the Fedora Project was finally ready to start distributing software packages.

The rules for package creation, package maintenance and quality assurance were clear and Fedora started producing *rpm*-packages; it was not long before some mirror servers were found which offered help in distributing Fedora. As everything in Fedora was well planned, there was of course the desire to structure the mirror system as well as possible. This is the point where this dissertation starts: discussing the development of a *Distributed Software Installation for Free Software*.

3. Initial Survey

As the goal is to develop something to support a community-based Free Software project, this dissertation will of course be developed as Free Software and also be based on Free Software as much as possible.

A very surprising and rather unanticipated change in Red Hat's product policy happened which led to the merger of the Fedora Project and Red Hat. Red Hat will no longer sell a cheap consumer-oriented Linux Distribution but will only offer an enterprise oriented Linux Distribution with the name Red Hat; parallel to this, it will develop a community-based Linux Distribution with the name Fedora [6]. Although this merger happened, the former Fedora Project will still continue to exist, but it will act as a filter through which new packages will enter the new Linux Distribution named Fedora.

The first and main part of the whole system needs to be a database in which every mirror maintainer enters the information about his mirror server and himself. The main reason for having information about the mirror maintainers as well is that, in some rare cases, it may be necessary to reach them if, for example, a very important security update is necessary or the mirror server has not been reachable over a longer period. This database will then be the basis of the other parts of the system.

An important part is the monitoring of the mirror servers to ensure that the list of available mirror servers is accurate and the user does not have to start searching for the best server. The monitoring should not only include the test that the mirror server is available and up-to-date but it should also verify the integrity of the repository. A possible way of testing the integrity is to use *gnupg* [7] to sign parts of the repository with a private key and the integrity can then be verified by the public key.

The monitoring of the mirror servers should lead to a list of servers which are reachable and up-to-date. This list is then the basis upon which each user can update his system or install new software. The user can select his preferred mirror server from this list. In addition to a manual selection of the best mirror server by the user, an automatic mode might select the best mirror server by measuring which server is the best for the current user. This can be achieved by measuring which of the available servers offers the best bandwidth usage or has the best reachability in terms of network distance.

Another part might be how to handle the initial installation of a front-end to the package manager which handles the dependency resolution. This largely de-

pends on whether the new Linux Distribution Fedora will include a tool like *apt* or something similar.

4. Aims and Objectives

During the initial survey, the following key parts of the system were identified:

- **Database** - As already mentioned, a database will be the central part of the whole system. The database will, of course, not be developed but used as a storage for the necessary data. The focus of this part will be to offer an interface to the database for the users and administrators.
- **Monitoring** - Based upon the entries in the database, another part of the system will monitor the mirror servers' availability and data integrity. This will be a periodic process which will be run every hour, for example.
- **Client Side** - The parts which are developed for the client side are still dependent on how the new Fedora Linux Distribution develops. On all accounts and independent of the distributions' development, a tool will be created which lets the user select which mirror server should be used for the software installation. This tool needs to offer a command line interface as well as a graphical user interface.

5. Initial Design

After the definition of the aims and objectives the following initial design has been created:

5.1. Database

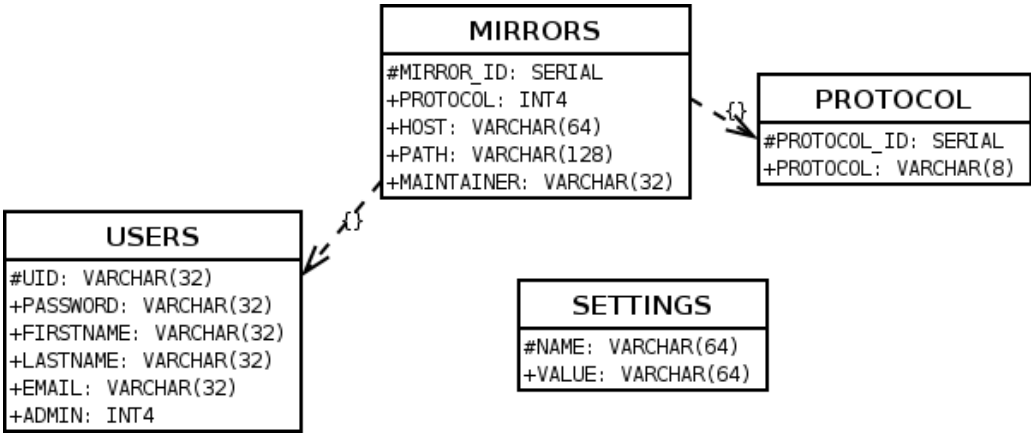
This part requires two steps. The first step is to determine a database design including all the necessary tables, the second step consists of the development of the user interface for the database. The design of the user interface depends on the database design and the database design depends on the user interface. As one aim was to offer a user interface which can be accessed from all over the world, the decision to offer a web-based interface was inevitable. This decision also influenced the database design as, with a web-based interface, it becomes necessary to offer user management. A user management indepent of the one from the system offering the web interface has the advantage that these users do not require full access to the system. This could, of course, be a security problem and therefore the user management is completely independent of the system it is running on. There is, however, also a disadvantage in using this design. It requires the user authentication part to be developed again although something like this has been developed many times before. As the data stored in this database are not really valuable, the authentication uses a simple design.

In order to secure not only the transmission of the user-name and password but also the transmission of all the data against eavesdropping the web-based part of the application should only be used with HTTPS[4]. This offers a basic security using the resources provided by web servers and browsers. To provide additional security for the passwords, these are not stored as clear text in the database but as a MD5[5] hash.

[4] Hyper Text Transfer Protocol Secure sockets

After these initial thoughts, it was decided to create the following tables:

Figure 1. Database Design



- **USERS** - This is the table containing the information about the users allowed to access this system.
- **MIRRORS** - This is the table where the data about each mirror server are stored. If a mirror has a maintainer, it will also be referenced in this table. Each maintainer can be responsible for multiple mirrors.
- **PROTOCOLS** - This table will be used to store the protocols the mirror servers provide. The content of this table will be probably be something like http, ftp and rsync.
- **SETTINGS** - This table is only used to store settings for the front-ends using this database. Possible entries are something like the e-mail address the

[5] Message-Digest Algorithm 5 is a message digest algorithm and cryptographic hash function with a 128-bit hash value.

system is using to send mails.

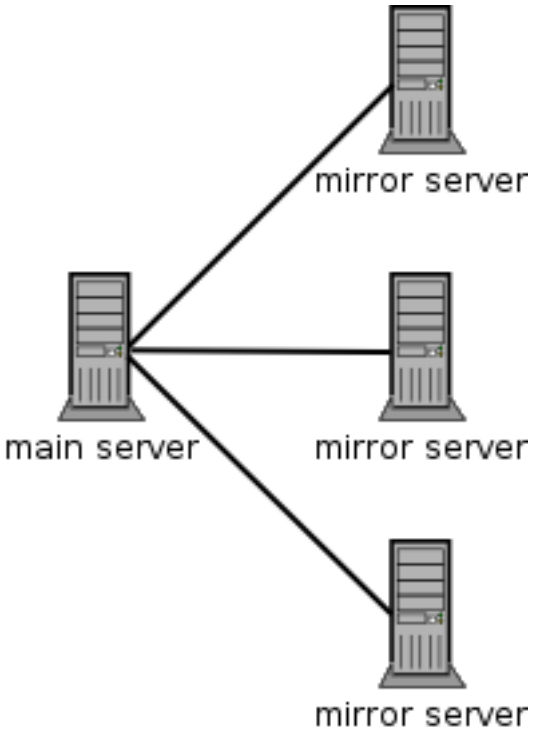
The user web interface will be written in PHP[6]. PHP is a general-purpose scripting language, but is also very well suited to embedding in HTML. This is one reason for using PHP, but the database support also makes PHP a very good choice. It is also the goal to develop a command-line front-end for some parts of the management of the data about the mirrors. The reason for a command-line interface is that it is much easier to add many new users to the system on the command-line than through a web interface, as a command-line tool can be used much better in scripting. It has not yet been decided whether this command-line interface to the database will also be written in PHP as this depends on the programming language used for the other parts of the whole system. Possible languages are Perl[9], Python[10] and PHP.

5.2. Monitoring

To monitor the available mirror servers, a connection will be established to each of these servers. During this connection, the main server will try to read a certain file from the mirror server. This file will contain a time-stamp from which the main server can determine when the mirror server has synced its data with the main server.

Figure 2. Monitoring

[6]PHP Hypertext Preprocessor[8]



The file containing the time-stamp will be generated at the main server at a certain interval (every 10 minutes, for example). A possible extension to the time-stamp file is to sign the file digitally as a check on the integrity of the mirror server. The problem with this approach is that the integrity check is only carried out for the file containing the time-stamp and therefore does not guarantee that the remaining data of the mirror server are still valid.

The check of each mirror server will probably be made every hour. This depends on the number of mirror servers available as it is important that the checking of the mirror servers should not require too many resources, and one hour seems a reasonable interval to check quite a large number of mirror servers. It is also important that the timeout after a mirror server is marked as unreachable is rather small so that not all the time necessary for checking the

availability is spent waiting for servers which are down. The feature to monitor the mirror servers every hour will not be part of the monitoring process but will be implemented by using the already-existing cron daemon.

Another idea for improving the performance of the monitoring is to parallelize the monitoring process. It is, however, important that the level of parallelization does not exceed a certain amount of processes so that the main server is still able to work normally and does not break down under the high workload.

The monitoring process will be developed using either Perl or Python as the programming language, but this still depends on the other components. At present, it is more likely that it will be developed using Python.

5.3. Client Side

The client side will consist of a tool connecting to the main server and requesting the list of available servers. The user then has to select one of the available mirrors from which the software should be downloaded from now on. The user can also use this tool to check which of the mirror servers is closest with the best connection for the user. The closest mirror will be the one with fewest network hops and the shortest time between icmp-echo-request and icmp-echo-reply.

At present, the goal is to offer a command-line version as well as a GUI[7] version of the client-side tool. The GUI version, however, will be developed first. During the design and development, this will be an important point so that the

[7] graphical user interface

logic of the tool will be separated from the view and it will thus be easy to replace the GUI-based view with a command-line based view if required.

6. Time Plan

- **01.10.2003** - start of Master's dissertation
- **15.10.2003** - first prototype developed for client-side application
- **31.10.2003** - database design and web front-end prototype finished
- **30.11.2003** - interim report and monitoring tools finished
- **20.12.2003** - development of all software components finished
- **31.01.2004** - Master's dissertation finished

7. Outcome

The expected outcome of the Master's dissertation is the analysis of a possible implementation of a *Distributed Software Installation for Free Software* as well as the implementation of a prototype offering the functionality described above.

References

Internet

[1] *Debian: http://www.debian.org/*. (November 2003).

[2] *Red Hat: http://www.redhat.com/*. (November 2003).

[3] *The GNU Project: http://www.gnu.org/*. (November 2003).

[4] *Conectiva: http://www.conectiva.com/*. (November 2003).

[5] *Fedora: http://www.fedora.us/*. (November 2003).

[6] *Fedora: http://fedora.redhat.com/*. (November 2003).

[7] *Gnupg: http://www.gnupg.org/*. (November 2003).

[8] *PHP: http://www.php.net/*. (November 2003).

[9] *Perl: http://www.perl.org/*. (November 2003).

[10] *Python: http://www.python.org/*. (November 2003).

Bibliography

Books

[Harlow99] Eric Harlow. Copyright © 1999 New Riders Publishing. 0-7357-0021-4. New Riders. *Developing Linux Applications*. with GTK+ and GDK.

[Bovet01] Daniel P. Bovet and Marco Cesati. Copyright © 2001 O'Reilly & Associates, Inc. 0-596-00002-2. O'Reilly. *Understanding the Linux Kernel*.

[McCarty99] Bill McCarty. Copyright © 1999 O'Reilly & Associates, Inc. 1-56592-705-2. O'Reilly. *Learning Debian GNU/Linux*.

[Wall00] Larry Wall, Tom Christiansen, and Jon Orwant. Copyright © 2000 O'Reilly & Associates, Inc. 0-596-00027-8. O'Reilly. *Programming Perl*. Third Edition.

[Pressman01] Roger S. Pressman. Copyright © 2001 The McGraw-Hill Companies, Inc.. 0-07-365578-3. McGraw-Hill. *Software engineering*. a practitioner's approach. Fifth Edition.

[Gamma95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Copyright © 1995 Addison Wesley. 0-201-63361-2. Addison Wesley. *Design Patterns*. Elements of Reusable Object-Oriented Software.

[Coulouris01] George Coulouris, Jean Dollimore, and Tim Kinderberg. Copyright © 2001 Pearson Education Limited. 0201-61918-0. Addison Wesley. *Distributed Systems*. Concepts and Design. Third Edition.