Department of Computer Science

Henrik Skantz
Mikael Böhm

Evaluation of log data for Wide Area Network Troubleshooting

Master's Thesis

2004:01

Evaluation of log data for Wide Area Network Troubleshooting

Henrik Skantz
Mikael Böhm

This thesis is submitted in partial fulfillment of the requirements for the Masters degree in Computer Science. All material in this thesis which is not my own work has been identified and no material is included for which a degree has previously been conferred.

Henrik Skantz

Mikael Böhm

Approved, 2004-02-18

Opponent: Johan Garcia

Advisor: Thijs Holleboom

Advisor KTHNOC: Bengt Gördén

Examiner: Donald F. Ross

Abstract

One of the common tasks for network administrators is to monitor routers and their error logs for abnormal behavior. Network administrators will on a daily basis investigate these logs for any errors protruding from the every day considered harmless errors. Identifying and attending to failures are most critical to the efficient operation of large computer networks.

In this master thesis troubleshooting assistance for **W**ide **A**rea **N**etworks was studied. Extended WAN troubleshooting in the personnel independent area was implemented by using network error logs. Error information is extracted from the logs that can be of interest in troubleshooting WANs. Investigations and development were made on network error log deviation, link failure, recurring error sequences, error diffusion and low frequent errors. With improved analysis on the historical data malfunctioning network components are easier detected and fixed, thus maintaining high network availability and efficiency. The development in this thesis produced algorithms that isolated certain network behaviors and events. The result is three algorithms that find diffusion, recurring sequences and low frequent errors in the network log data. A theory for link failure detection is presented but not implmented.

Contents

1 Introduction **1**

1.1 Research network 2

1.2 Framework overview of network surveillance 2

1.3 Thesis overview 5

1.4 Summary 7

2 Background and Theory **9**

2.1 Introduction to WAN troubleshooting 9

2.2 Evaluation of network log data errors 10

2.2.1 Base system 10

2.2.2 Log data information 11

2.2.3 Data set theory 14

2.2.4 Related work 15

2.3 Project outline 15

2.3.1 Diffusion error detection 16

2.3.2 Recurring sequences 20

2.3.3 Low frequent errors 26

2.3.4 Network log deviation 33

2.3.5 Link failure detection 36

2.4 Summary 44

3 Experiment and Implementation **45**

3.1 Implementation objectives 45

3.2 Network environment 46

3.3 Implemented algorithms 47

3.3.1 Diffusion error detection 47

3.3.2 Recurring sequences 53

3.3.3 Low frequency errors 61

3.4 Summary 68

4 Results and Evaluation **69**

4.1 Algorithm test outline 69

4.2 Test environment 70

4.3 Algorithm result and performance 70

4.3.1 Diffusion error detection 70

4.3.2 Recurring sequences 75

4.3.3 Low frequent errors 80

4.4 Algorithm evaluation 87

4.5 Summary and information 89

5 Conclusion **91**

5.1 Thesis conclusion and evaluation 91

5.2 Problem areas 93

5.3 Future work 94

5.4 Algorithm results from KTHNOC 95

References **97**

A Diffusion files **99**

A.1 diffusion.c 99

A.2 diffusion.h 111

B Recurring sequence files **115**

B.1 rekerr.c 115

B.2 rekerr.h 130

B.3 rekerrMySQL.c 134

B.4 rekerrMySQL.h 136

C Low frequent files **137**

C.1 lowfreq.c 137

C.2 lowfreq.h 150

D Common files **157**

D.1 timeCommon.c 157

D.2 timeCommon.h 157

D.3 readconf.c 157

D.4 readconf.h 159

D.5 listCommon.c 160

D.6 listCommon.h 163

D.7 filectrl.c 165

D.8 filectrl.h 167

D.9 mysqlQ.c 168

D.10 mysqlQ.h 174

D.11 makefile 175

E Script files **177**

E.1 lowfreq.sh 177

List of Figures

1.1 GigaSunet core network logical layout over Sweden 3

1.2 Framework overview of network management. 4

2.1 Base system schematic overview. 11

2.2 Example of one log data row. 13

2.3 Network error spread 19

2.4 CPUHOG error diffusion in network 20

2.5 Random output cut from mysql router index and error code 23

2.6 Illustrated an example of a sequence of error events. 25

2.7 Example of low frequent error irregularity. 30

2.8 Output from *mysql* database. 31

2.9 Schematic view of connectivity from part of network. 37

2.10 Router interface *Link State Machine*. 39

2.11 Output from log database. 40

3.1 Base system schematic overview with new component 46

3.2 Assumed error diffusion in network 48

3.3 Error diffusion plot 2003-02-26, 14:40:00 - 14:50:00 49

3.4 *time-window* 51

3.5 New position of the *time-window* 51

3.6 Zoomed time-window. 52

3.7 The same diffusion detected twice 53

3.8 Diffusion overlapping two time-windows 53

3.9 A missed diffusion, with a reportlevel set to 5 53

3.10 Case A: {1A, 2B, 3B, 4C, 1A} Not managed, Source not reliable. 55

3.11 Case B: Managed if 1A is the starting node. 56

3.12 Case C: Managed, 1A and 2B can be starting nodes; *seqlen* 4 or 3. 56

3.13 A recurring sequence, a pattern is found. 57

3.14 List in which patterns are found. 58

3.15 Three scenarios when searching for patterns. 60

3.16 Cut from low frequent algorithm data storage structure. 62

3.17 Error frequency areas. 64

4.1 Output from the diffusion program. 72

4.2 SQL query on the interval 05:28:25 - 05:28:35. 73

4.3 SQL output, time-window 05:28:30 - 05:28:40. 74

4.4 Diffusion summary window. 74

4.5 Program output run from a console 77

4.6 Using the GUI to retrieve more information about an error. 78

4.7 Cut of program output with *show all* option 79

4.8 First occurrence for pattern [1]. 80

4.9 Last occurrence for the first pattern. 81

4.10 Low frequent entry printout. 82

4.11 Low frequent time period representation. 84

4.12 Low frequent error result graph. 85

4.13 Low frequent program output summary. 86

5.1 Example of a trend graph. 95

List of Tables

2.1 Table showing a fictitious database table with necessary data for knowledge about network link connectivity. 38

2.2 Table showing the probability for link failures for state machines of router interfaces and their common physical link. 41

2.3 Table showing the state machine events for figure 2.11. 42

3.1 Diffusion 51

4.1 Test results for May, June and July. 71

4.2 Recurring sequence test runs results 75

4.3 Low frequent example results. 83

Chapter 1

Introduction

Network operators face the difficult task of maintaining the network quality and at the same time introducing new technologies such as software upgrades and new hardware. Troubleshooting involves a lot of manual work and requires detailed knowledge, and is often time consuming and difficult to plan. Often the amount of data to analyze is far too great for human attention. The relevant data for problem detection also is sometimes scattered and has to be manually gathered.

There are also economical aspects to fault detection. Network links are expensive to hire[1] and any detection of a link failure will reduce costs with a fine in an agreement between the parties. Automated link failure detection is therefore not only of quality but also of financial interest.

Traffic anomalies such as failures and attacks are commonplace in today's computer wide area networks. Identifying, diagnosing and fixing anomalies are essential parts of every day network administration, without which networks would neither operate efficiently, nor reliably. Accurate identification and diagnosis of anomalies depends on robust and

[1]Network operators often hire links. Costs are based on the link capacity among other things.

accurate data, and further, on established methods for isolating anomalous signals within that data.

1.1 Research network

The research presented here has been done in cooperation with KTHNOC[2]. They have been maintaining the Swedish University Computer Network, called SUNET, since its startup 1980.

The backbone network is called GigaSunet and its structure is shown in figure 1.1. It is one of the world's fastest university networks, operating at 10 Gigabit per second. This network is the base for this research. It has all the needed properties to be a suitable research object. It is complex, large, modern and has high demands on quality.

To maintain quality in a network surveillance tools are generally used. A lot of the present network surveillance tools are expensive and still do not perform any diagnosis based on historical events. There are also free tools released under the GPL [4] license such as Ntop [6] that are well functioning for realtime surveillance, but require human attention at operation and are therefore not always cost effective.

1.2 Framework overview of network surveillance

The network surveillance is dependent on well functioning fault detection and diagnosis methods. Network management can be viewed in a framework overview to distinguish different troubleshooting areas. Such a layout is shown in figure 1.2. As the figure shows, most common are parts that involves personnel dependent management.

[2]Kungliga Tekniska Högskolan Network Operation Centre (KTH *eng*: Royal Institute of Technology). http://www.noc.kth.se

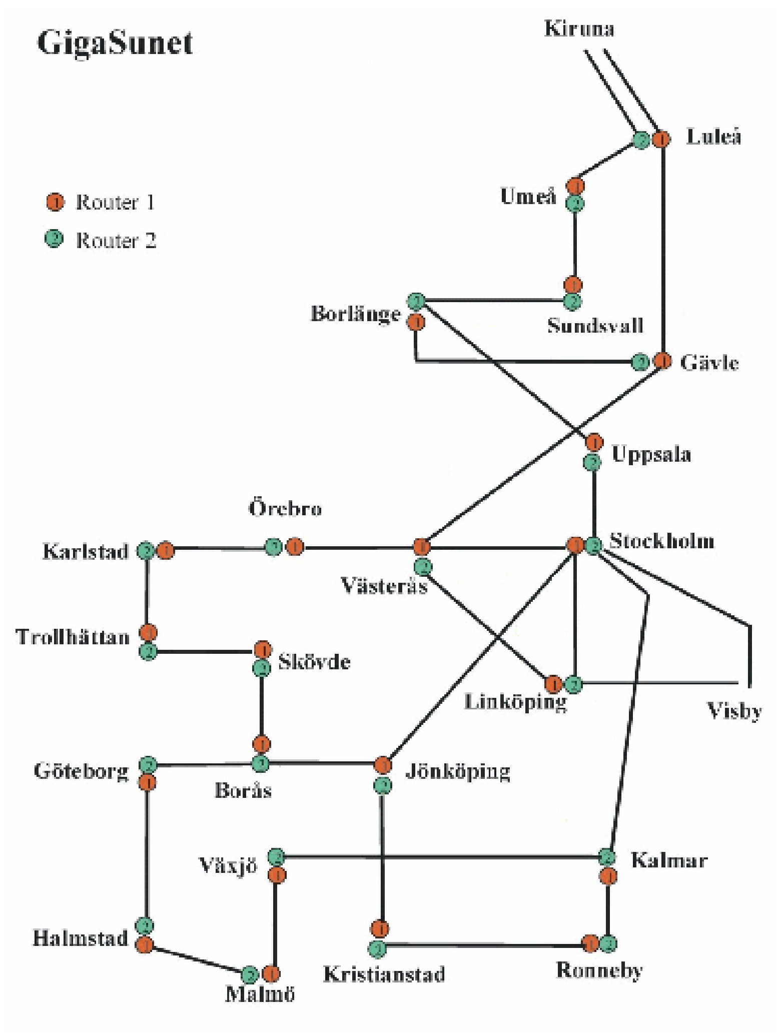


Figure 1.1: GigaSunet core network logical layout over Sweden
The data used in this project are archived log data from the network. This figure is modified from SUNET web page, http://www.sunet.se.

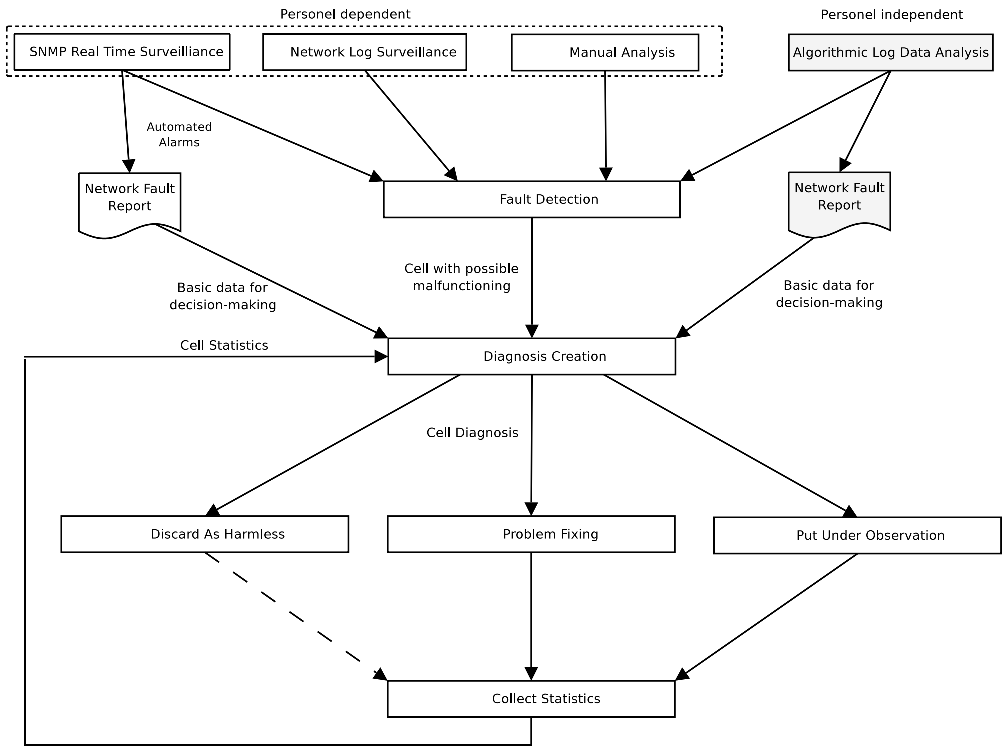


Figure 1.2: Framework overview of network management.

The shaded parts shows the extended personnel independent part of network surveillance.

This research will try to extend the entire structure with a new area of troubleshooting Wide Area Networks and with a minimum of extended staffing requirements. The shaded part in figure 1.2 shows the extended part of network surveillance in the framework model. The goal is to produce automated report basis for the network diagnosis process.

As figure 1.2 shows, we have two positive aspects involved extending the quality of network management:

- **Fault detection**. Finding unknown malfunctioning cells.
  (Algorithmic log data analysis)

- **Diagnosis creation**. Generated reports with cells and errors.
  (Network fault report)

Based on the content analysis from technical expertise, processed events and reports follow different paths and actions.

1.3 Thesis overview

This thesis cover some research areas as well as implementations of ideas into algorithms written in C. The structure of this thesis is as follows:

Background and theory Will give background and theory for network log data investigation. Chapter 2.1 introduce some basics in WAN troubleshooting and information about the network log data and how this data is handled by a database system. Chapter 2.1 also give information about this project research network, GigaSunet.

The network log data will be formalized in mathematical set theory in chapter 2.2.3. This will later on be used further for each problem area as a base for developing theories.

A project outline in chapter 2.3 present an overview of this thesis research areas. Certain network error states that are hard to discover are outlined for investigation. Selected areas are diffusion, recurring sequences, low frequent errors, log intensity deviation and link failures. They are discussed and formalized with data set theory in chapters 2.3.1, 2.3.2, 2.3.3, 2.3.4 and 2.3.5 for each area respectively.

Experiment and implementation This chapter discuss details about the implemented problem areas. Some general objectives about implementations ideas are given in chapter 3. Facts about the network environment and how data is collected for analysis is given briefly in chapter 3.2.

Problem solving algorithms are each presented separately in subareas as in chapter 2.1. Implementation details for the error diffusion algorithm is given in chapter 3.3.1. Recurring sequences in chapter 3.3.2 and the low frequent implementation in chapter 3.3.3.

All implementations are constructed from the earlier given theory in chapter 2.1. The implemented algorithms will only show one possible way to find error states in the network. They all have their flaws but are in first place meant to verify the theory developed in this thesis.

Results and evaluation The results for implemented areas are discussed and their technical functioning evaluated. Error diffusion is evaluated in chapter 4.3.1. Found diffusions are verified and analysed for integrity against the data in the database. Algorithm reported data is shown and test results are presented in tables with different algorithm input parameters.

Recurring sequences is discussed in chapter 4.3.2 and low frequent errors in chapter 4.3.3. They are evaluated as in the diffusion case. Real cases are shown but no significance in their meaning in fault management is performed. This require a network technicians skill and is not a subject for this thesis. The implemented algorithms functional behavior, flaws and limitations are discussed in chapter 4.4. This chapter also suggests how these algorithm might be used on a daily basis for optimal performance.

Conclusion The last chapter 5.1 summarizes the thesis and gives overall project conclusions. The chapter discusses weaknesses and the potential of the different areas of network log data analysis. Problems evolving in this thesis are specifically gathered in chapter 5.2. The ideas for future development and use of this work can be found in chapter 5.3. The chapter focus on the link failure detection problem which does not have a test implementation. But also how to better handle the algorithm output data. The low frequent algorithm has proved some usefulness during tests. Some details about this concludes this thesis in chapter 5.4.

As described above, this thesis considers areas of pure theory in some parts, but does also contain more straight forward implementations of known existing error states in the research network. But a theory has been outlined for all concerned areas, even the more straight forward implemented ones.

1.4 Summary

This research concentrates on analyzing the errors occurring in networks based on historical data. Many tools have been developed to automatically generate alerts to failures, but generally it has been difficult to automate the anomaly identification process. We investigate new methods for troubleshooting assistance based on the continuous log data generated from the network components. We present ideas for generating diagnosis reports based on algorithms performed on this data. To extend the troubleshooting assistance captured from the data we make several assumptions and investigate if they can produce results of practical interest.

Chapter 2

Background and Theory

2.1 Introduction to WAN troubleshooting

As a network evolves and becomes larger and more complex the need for extended fault identification and diagnosing increases. An error in a network can affect the network in many different ways. For example a single error can propagate and develop into an error state. By error state we mean that a part or the entire network is non functional. Single errors can result in a larger number of errors creating redundant information that conceal the actual error. This research will focus on finding error indications as well as their origin.

In GigaSunet, which we are considering here, log data is constantly generated from the backbone network routers. All incoming data is archived and saved for future analysis and scrutinizing. New incoming data is inspected manually for understanding of possible reasons for malfunctioning in the network. This paper will focus on development of tools for helping network technicians in their daily work. The expertise of the operators is still required for drawing conclusions, doing repairs and applying countermeasures before possible traffic disturbances occur.

2.2 Evaluation of network log data errors

Our research focuses on extracting more information from the log data generated by the network routers. In collaboration with the network technicians at KTHNOC we identified three types of errors that may need a more detailed analysis. These types are rapidly spreading errors, recurring errors, and low frequent errors. These will be discussed in section 2.3.

Mathematically we have a large set of elements, each element representing a network event. All elements in this set are unique, characterized by their different attributes. A significant amount of those are just to be considered network noise, but they contribute to the complexity and make human analysis harder. By applying filters, mainly following mathematical set theory, most errors and their relations can be found, but only if one knows what to look for. We need to have a tool that can look at some selected attributes of each element and search the whole set for relations or patterns among those attributes. If such a match can be found then we can reveal and trace an error state inside the set.

We approach the task by defining algorithms for each problem area that we will investigate. Selected areas will be evaluated with practical testing. Code is written in the C programming language to implement the algorithms and ideas for test runs on the research data.

2.2.1 Base system

In our earlier work [1] we developed the base system making this research possible. Previous to our system all data was kept in plain text files making it hard to analyse effectively. The base system parses every incoming log row and inserts it into a relational database system. The database is organized in third normal form, which provides possibilities to run algorithms efficiently on the data.

The graphical user interface (GUI) which is a part of the base system has made it possible to easily study the data stored in the database. The GUI has some basic features to make quick selections and filtering of data. A system outline is presented in figure 2.1. A process running an analyzing algorithm can connect to the database shown in this figure and select the required data suitable for the required job.

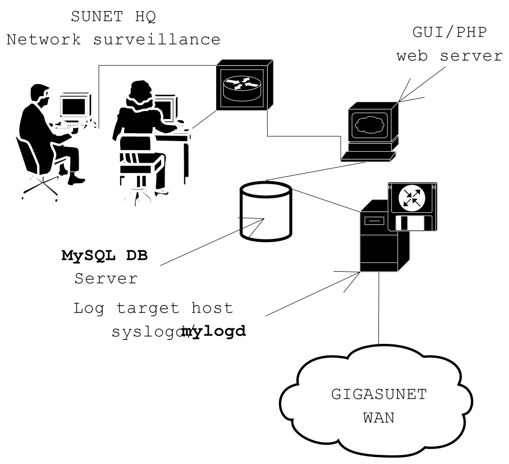


Figure 2.1: Base system schematic overview.
All error logs are put in a database. The Mylogd host receives all error logs from GigaSunet and stores them in a database. Data stored in the database is accessed by a web interface.

There are always possibilities to develop more features for the database user interface. These are not considered in this paper, still these can also facilitate troubleshooting on a daily basis. But they are limited to being smart filters, adapted user interface etc. For the interested reader we recommend [1].

2.2.2 Log data information

The network routers log data is the base for this research project. Hence, it is essential to be familiar with their format and the information they contain since extracting more information from this data is the main task. This implies to look for special types of log rows and locating relations in their mutual appearance in time. Network log timestamps gives the opportunity to sort events in chronological order. This is essential for most theories in this thesis as will be shown later on in this chapter.

Log row format

It is necessary to study how the plain text row of log data is formatted and treated as an element in the database. Basically every event produces one row of data providing the following information:

- Event archiver time
  - This represents the time when the data arrived at the host collecting the data. It is the time stamp set by the syslog daemon. The host running the archiving daemon is often time synchronized with NTP[1].
- Event location
  - Location of the event. The router is often represented with its hostname or IP number.
- Event errorcode
  - All events have defined errorcodes. There are at least six thousand different codes with error severity from zero to six, six being the lowest.
- Event component time stamp
  - All routers add a local time stamp to an event. These are also often synchronized with the NTP system.
- Event process identifier
  - All processes have as in common UNIX systems a process identifier number called PID.
- Event message

[1]Network Time Protocol. See: http://www.ntp.org for more information

```
May 15 22:35:58 stockholm4.sunet.se 87814: May 15 22:35:57.098: %BGP-3-NOTIFICATION:
        sent to neighbor 194.68.128.26 2/2 (peer in wrong AS) 2 bytes 0758
```

Figure 2.2: Example of one log data row.

– This part of the row is additional information for an event. On different modes in the router this could supply memory status, network interface down/up etc.

These are the actual attributes of each element in the set of events. A generated plain text row containing those attributes is shown in figure 2.2. The row is parsed and inserted into the database by the base system daemon called **mylogd** in figure 2.1.

Every event in the network routers generates one row of data. During this research the GigaSunet network has generated approximately one row of data every two to tree seconds. This is valid at normal state of operation around the clock. At reconfiguration, or error states, it could easily rise to thirty rows every second. Hence, manual inspection is hard work even with UNIX tools such as *sed*, *grep*, *sort* and *awk*.

Present log data processing

By observing the log data network technicians can draw conclusions about the network status and locate failures. Still it is not possible to diagnose all failures, or even find them, since the amount of log data simply is too large.

To investigate the network status the technicians can apply filters and sorting on raw log data kept in plain text. This is often done with the UNIX tools as mentioned before. The tools can filter the dataset and pipe a subset to a new filter and so on. But even with the use of UNIX tools in an intelligent manner, there are limitations to what they can accomplish on the log data. And further, these tools are not intended for use with a database system[2] output.

The standard UNIX tools can in principle be used to look for, e.g a recurring pattern. This would however require extensive programming in for example the AWK scripting

[2]The database system referred to was earlier explained in chapter 2.1

language. This would result in a slow processing of the data. The use of a full fledged and efficient compiled programming language like C is a more natural solution.

It is also essential to minimize the workload on the database during a network error investigation. The main task of the database is to store log data and to have this information at hand, not to be an active component in demanding data processing.

2.2.3 Data set theory

The filtering of log data can be described in terms of mathematical set theory. In this section we explain this relationship formally, in order to better understand some of the limitations that are involved.

If we let ∂ represent the set of log data, A is the set of all routers in Karlstad, B is a specific router in Stockholm, C is the set of all routers with errorcode X. To inspect all routers in Karlstad together with a specific router in Stockholm, and further, we do not want to see any error with type X, a simple filtering procedure for a subset S could be applied like this:

$$S = \partial \cap (A \cup B) \cap \neg C$$

It is obvious that all derived sets S are subsets of ∂, i.e,

$$S \subseteq \partial$$

Often the obtained subset S is far too large for human inspection. Today all subsets are derived manually or semi-automatic from the user interface by the base system. The effectiveness of this procedure very much relies on the operator's skill, experience and intuition.

Also it could require too much work to be a realistic task for a certain investigation. For example, it is obvious that the set theory does not provide a tool to give us *the least occurring* members in the origin set. It would require counting the members of all possible subsets derived from filtering for all known errorcodes. And errorcodes are often counted in thousands.

2.2.4 Related work

Network fault detection methods have become more intensely studied over the years. Network traffic properties have been studied for many years - examples of typical traffic analysis can be found in [10], [12] and [13].

Our focus is in identifying certain characteristics of archived log data captured from the network. Prior studies on network fault detection, [2], [3] and [5] use statistical deviations from normal traffic behavior to identify faults. In [7] and [8] anomalies in the network are identified and characterized based on SNMP[3] and IP flow data.

2.3 Project outline

The project outline ideas emerged from interviews with technical staff and by observing real case error states. The project focus on error states which are hard to find and therefore might go unnoticed. Some are likely to exist but they are impossible to survey and put in coherence from the total amount of data produced. This project focus on the data stored in the database and what possible information it can bring.

The issues of the project can be summarized as follows:

- i Detect diffusion errors. Errors that spread rapidly over the network.
- ii Detect recurring sequences of errors.
- iii Find low frequent errors and their coherence over long periods of time.

[3]Simple Network Management Protocol

iv Detect deviations from normal state of network operation, detecting increase or decrease of log data.

v Link failure detection.

By studying historical log data carefully, there are reasons to believe that all events represented by the items above, are more or less likely to occur. As explained in chapter 2.2.2 and 2.2.3, applying filters is not enough to find these relations in the data or an immense task. More detailed background information will be provided in the following sections for each area respectively.

2.3.1 Diffusion error detection

If a certain error in one of the network routers instantly causes a chain reaction of the same error in its neighboring routers, we are facing what we call a diffusion error. That is, the error will spread within the network in a short period of time. It might pass without notice since the times involved often are less than five seconds. It might only cause a short interrupt in the network and therefore not cause any user to react or come to some technicians attention. Still these errors are considered to be serious and always need to be analyzed. Such errors can, if they start to repeat in short intervals or change their behavior to act recursively, cause serious traffic disturbances.

Diffusion set theory

We define network error diffusion as:

One or more identical errors that within a limited period of time are reported from two or more network components

Formally we define error diffusion when:

the attribute error in a set has the the same value for all routers in the set and the attribute timestamp belongs to a limited period of time, and further, the number of members in this set exceeds one.

By applying set theory we can define diffusion as follows:

Let S be a certain set of errors, $E(e)$ represent an element **error** attribute and $T(e)$ represent an element **time** attribute.

For all error elements e it holds that they all belong to the set of errors, that is

$$\forall e \ [e \in S]$$

We let A represent a specific error attribute and T a time window as:

$$T = \{t | t_0 < t < t_0 + lim\}$$

where t_0 is an observed time and lim is a constant.

If the following holds:

$$D = \{e | E(e) = A \wedge T(e) \in T\}$$
$$\text{for all } e \in S \text{ and}$$
$$\mid D \mid > 1$$

we have a diffusion.

Diffusion algorithmic variables

There are two numbers to laborate with for adjustment to better suite the operating environment. The first is lim, the time to look for diffusion. We can say that this represents the *time window size.* Secondly, we can increase the *minimum number of elements* in a set for being defined as diffusion. From the minimum of two elements up to a number equal to the actual number of routers in the network.

From previous experience in our research network[4], the time period *lim* should reasonably be somewhere around ten seconds. This depends on the capacity, size and speed of the network. For our research environment with about fifty Cisco 12410[5] core routers, the time variable *lim* set to ten seconds will do as a good reference to start with.

The high speed of the network will let an error spread to all routers within two to three seconds, but we also have to consider some processing delay at each router depending on the nature of the error.

The number of routers within the time interval need to be more than two in practice to be defined as diffusion. The explanation is that there are common irrelevant errors we consider to be network noise. Those errors will otherwise report false diffusion errors by coincidence of matching noise.

The algorithm will need to have those numbers as input at startup. They will in practice be typical in arguments to the program.

Diffusion case study

To see how diffusion could show up in practice we will study a real case. On the 24 of February 2003 there was an error diffusion in SUNET. Figure 2.3 shows a representative part of KTHNOC's log capturing system [9] of the two first seconds from the incident.

[4]The research network layout is shown in figure 1.1 in chapter 1. More information on the on the GigaSunet network are available at: http://proj.sunet.se/gs/

[5]For product information and technical data visit: http://www.cisco.com

```
router name           error          time     date
...
halmstad2.sunet.se    %SYS-3-CPUHOG  10:17:10 2003-02-24
orebro1.sunet.se      %SYS-3-CPUHOG  10:17:10 2003-02-24
orebro1.sunet.se      %SYS-3-CPUHOG  10:17:10 2003-02-24
uppsala1.sunet.se     %SYS-3-CPUHOG  10:17:10 2003-02-24
boras2.sunet.se       %SYS-3-CPUHOG  10:17:10 2003-02-24
borlange1.sunet.se    %SYS-3-CPUHOG  10:17:10 2003-02-24
gavle1.sunet.se       %SYS-3-CPUHOG  10:17:10 2003-02-24
skovde2.sunet.se      %SYS-3-CPUHOG  10:17:10 2003-02-24
linkoping1.sunet.se   %SYS-3-CPUHOG  10:17:11 2003-02-24
trollhattan2.sunet.se %SYS-3-CPUHOG  10:17:11 2003-02-24
umea1.sunet.se        %SYS-3-CPUHOG  10:17:11 2003-02-24
jonkoping1.sunet.se   %SYS-3-CPUHOG  10:17:11 2003-02-24
karlstad2.sunet.se    %SYS-3-CPUHOG  10:17:11 2003-02-24
orebro1.sunet.se      %SYS-3-CPUHOG  10:17:11 2003-02-24
boras1.sunet.se       %SYS-3-CPUHOG  10:17:11 2003-02-24
stockholm1.sunet.se   %SYS-3-CPUHOG  10:17:11 2003-02-24
borlange1.sunet.se    %SYS-3-CPUHOG  10:17:11 2003-02-24
borlange1.sunet.se    %SYS-3-CPUHOG  10:17:11 2003-02-24
gavle1.sunet.se       %SYS-3-CPUHOG  10:17:11 2003-02-24
skovde1.sunet.se      %SYS-3-CPUHOG  10:17:11 2003-02-24
skovde2.sunet.se      %SYS-3-CPUHOG  10:17:11 2003-02-24
...
```

Figure 2.3: Network error spread

The figure shows fifteen different routers within this time slot. Clearly in this case a ten seconds time window for this type of WANs will capture all essential diffusion taking place.

Every row in the figure represents an element in the sets discussed earlier. The four columns in the table are router name, error, time and date which the error occurred. The time and date are set by the central syslog daemon shown in figure 2.1 as the messages arrived at the host.

Figure 2.3 shows how the error %SYS-3-CPUHOG spreads rapidly over the network routers. That is, the same error is suddenly present in many routers when looking in a narrow time slot, hence we have diffusion. There could be numerous of different malfunctions causing an error spread, for instance software bugs or malicious attacks on the routers.

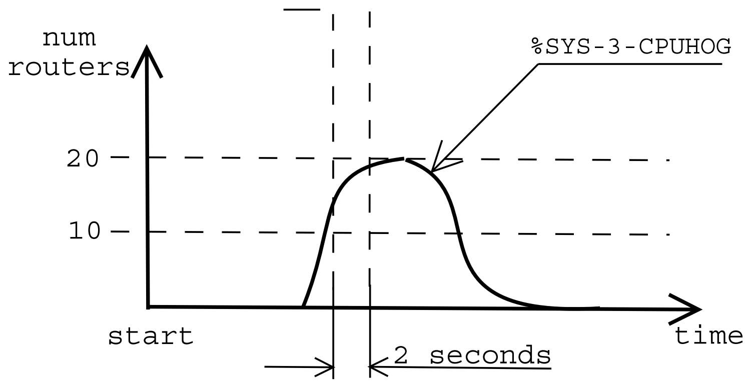


Figure 2.4: CPUHOG error diffusion in network
An expected graphical view of the errors' simultaneous presence in routers during the incident.

In this specific case the error %SYS-3-CPUHOG indicates that a certain process has run for a too long time without relinquishing the processor. The process is therefore a "CPU HOG". Before the process was killed by the router operating system, the indata causing the process to loop was forwarded to a neighboring router in the network and causing the chain reaction.

If we count the number of unique routers present in a two seconds' sliding time window over the incident, it is possible to plot a graphical view from the result. While moving the window unique routers are counted containing this specific error. An expected graph of the %SYS-3-CPUHOG incident from figure 2.3 is shown in figure 2.4.

2.3.2 Recurring sequences

The network connectivity will often generate a chain of events, an unwanted scenario seen in section 2.3.1. As shown before, a malfunctioning component can cause the neighboring routers to generate new errors. If a chain of errors occurs repeatedly, and if their mutual appearance are in the same chronological order, the recognition of the pattern can be used as a method to trace the error cause. The main idea is to find the first error in the repeating chain. Presuming there is a relation found, a removal of the first error ought to stop the chain reaction. However, to make a solid appraisal of each situation, it will always require a network technician's expertice. Hence, our main goal is to find an event and not draw any conclusion of its appearance.

One challenge in this area is to separate possible repeating sequences of errors from the network's randomized generation of log data. Repeating patterns might be reported by coincidence and it will unfortunately gain from the always present network message noise. There can be no guarantees not to report an irrelevant pattern generated by coincidence. Statistically that ought to happen since there is a limited amount of combinations with a large amount of data. This calls for the need of algorithm input variables to adjust to the environment to minimize the risk of reporting irrelevant repeating errors.

As in the diffusion case, assumptions are made of the existence of chain reactions between the errors, one error causes another and so on. By observing raw data one can not say much about the presumptive result and detailed algorithm requirements. This has to be discussed during testing. However, we can assume that it will be absolutely essential to construct an algorithm that focuses on the origin of the chain reaction. A network technician will only need to know the source of a recurring sequence to be able to start an error state investigation.

Recurring error theory

A recurring sequence is defined as:

when two or more router/error combinations appear in a chronological order, and their mutual order is repeated one or more times, and the components in that sequence also are *repeating exactly the same error sequence.*

Hence, synchronized time among the network components is absolutely necessary when searching for a repeating sequence to be able to make conclusions of the result.

In practice the formal definition is too rough to process in an algorithm without further restrictions. Adjustable input parameters for that purpose will be needed for requiring more than two components in a sequence and a requirement for more repetitions than one.

Case study

This case study will focus on two attributes from the log row elements. They are the *router name* and its corresponding *error type*. The router name is represented with an index number[6] in the relational database. A snapshot from the database with the relevant attributes is shown in figure 2.5. The figure shows only fifteen rows from GigaSunets March 2003 cisco table from a total amount of 296690 rows available from this month.

It is important to recall that there is a need for synchronized time with NTP to be able to have the database system to deliver its output data sorted by time to the algorithm. Otherwise the event order is unknown and patterns become meaningless to study. They will then only occur by coincidence. If NTP is not available, the event order in the network is similar to the event arrival at the host providing the syslog daemon. Accuracy will then depend on the event message transport time in the network from its different components.

If we take a closer look at figure 2.5 we can observe a recurring sequence. Starting at row tree we find the number sequence $\{2, 1, 3, 3, 7\}$ and a recurrence starting at row eight. We will only consider it to be valid if all elements in the sequence by definition have matching error codes with a repeating sequence. In this example they matched and we

[6] Index number i mapped against a table with router names to avoid redundancy. See [1] for additional information.

```
          mysql> select rindex,errorcode from ‘cisco-0303‘ limit 15;
                       +--------+------------------------+
                       | rindex | errorcode              |
                       +--------+------------------------+
                 1.    |      3 | %TCP-6-BADAUTH         |
                 2.    |      3 | %TCP-6-BADAUTH         |
          +-->   3.    |      2 | %PIM-6-INVALID_RP_JOIN |
          |      4.    |      1 | %PIM-6-INVALID_RP_JOIN |
      Seqv. (1)  5.    |      3 | %TCP-6-BADAUTH         |
          |      6.    |      3 | %TCP-6-BADAUTH         |
          +-->   7.    |      7 | %PIM-6-INVALID_RP_JOIN |
          +-->   8.    |      2 | %PIM-6-INVALID_RP_JOIN |
          |      9.    |      1 | %PIM-6-INVALID_RP_JOIN |
      Seqv. (2) 10.    |      3 | %TCP-6-BADAUTH         |
          |     11.    |      3 | %TCP-6-BADAUTH         |
          +--> 12.     |      7 | %PIM-6-INVALID_RP_JOIN |
                13.    |      2 | %PIM-6-INVALID_RP_JOIN |
                14.    |      1 | %PIM-6-INVALID_RP_JOIN |
                15.    |      3 | %TCP-6-BADAUTH         |
                       +--------+------------------------+
                       15 rows in set (0.00 sec)
```

Figure 2.5: Random output cut from mysql router index and error code
Note, the two sequences do not need to be connected to each other, noise can exist in between.

might have a recurring error of some sort.

Algorithm issues

There are some criterions we have to consider before an algorithm triggers a report. We will need to observe the following:

- How many repetitions of the sequence should be required.
- Minimum length of sequence pattern.
- How to avoid reporting a subset of an error sequence.

In practice the operator will need to laborate with input parameters to narrow the output to what might be of interest. A technician will always need to experiment from case to case when processing the data, depending on the network layout and status. Those input parameters will be the number of repetitions required and the minimum length of a sequence.

Avoiding subsequences will be equivalent to finding the component (event) that is causing the error sequence. Earlier showed in figure 2.5, we saw a repetition of an error sequence in the database table. The graph in figure 2.6 illustrates a sequence chain of ten steps of this kind, corresponding to ten rows in the database table. This example event sequence starts at **A**. But as soon as the first event **A** is found again, the sequence chain needs to stop. This will increase the chances of focusing on the chain generator. Hence, we define this chain to be of length seven stopping at **B**. The search for equal patterns will be processed for the first seven members in this chain starting at **A**, if seven is in the range of the operators minimum requirements.

Further, if matching patterns of this sequence are found, it will be necessary to move forward the *sequence length* on the next search to avoid reporting subsets of this particular

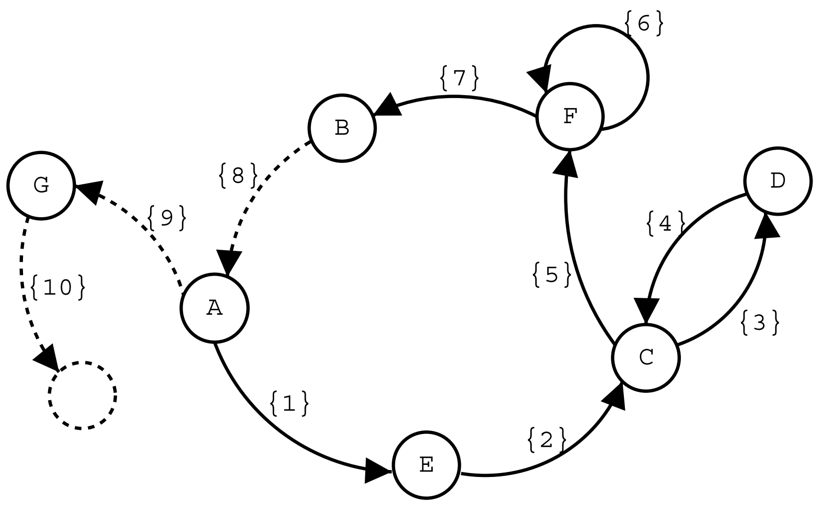


Figure 2.6: Illustrated an example of a sequence of error events.
In reality capital letters A - G represent a network component together with a specific error event.

sequence. In the example given in figure 2.6, the next search will once more start with event **A**, and continue with **G** as next and so forth to build up the next pattern. If not, reports could be generated without the actual cause of the event since subsets are presumed not to begin with, or even contain, the origin of the sequence.

One must also consider that a sequence can be broken by coincidence from a random error event. Still, if an error state remains for some time, enough repetitions are likely to be recorded without disturbance to have enough data intact to trigger a report.

As shown, this part of the research will comprise only to find repeating patterns. No significance of network connectivity and/or error types are considered. A closer look at figure 2.6 also reveals two loops. Internal loops are allowed since this research does not attempt to answer how relevant sequences should be formed. But allowing internal loops within patterns, as shown at **F** and **C-D** in figure 2.6, is an object of further discussion and experiment.

2.3.3 Low frequent errors

In a WAN like GigaSunet, error noise is common and KTHNOC usually treats these errors as not being harmful. Often these errors origin from ISP[7] networks that are connected to GigaSunet. Network technicians try to filter out as much noise as possible when they search for errors in the network log data. Still, they are often left with a lot of data to observe. What is crucial here is that some errors with importance for fault detection, might be hidden in the data, if they are rarely reported. We call these errors *low frequent* errors.

A low frequent error might only occur one to five times a month. This means in practice for GigaSunet, a rate of once a hundred thousand to once a million, or even less. Understanding this, no further explanations are required to realize why they are easily missed.

Low frequent errors could be possible time bombs that in the future might cause major failures. Especially if the grade of severity is high[8], it could be a failure that eventually shuts down a complete network router when reaching a critical limit or state.

The peak of the error rate may be very low, but this peak could be recurrent over some period of time. If the number of a specific error increases over a long period of time, and its increase rate is not high, it is hard to see this increase, or worse, not even notice the error at all. Hence, a malfunctioning network router might be overlooked until it causes serious network disturbance.

This provides two interesting aspects of low frequent error detection:

[7]Internet Service Provider

[8]If the grade of severity is high, the nature of the error is considered to be more serious by Cisco technicians. The grades are from 0 to 6 where 0 is the most serious type. The plain text error log generated by the router provides this data within the message as; %GROUP-[0-6]-TYPE.

- Errors that are rare but that have an increasing trend.
- Overlooked errors of serious nature.

At this point this project will not evaluate trend analysis. In this first phase, while producing low frequent error data reports, this data will have to be diagnosed by a network technician. Extended possibilities will be discussed in chapter 5.3.

From discussions with technical staff there are reasons to believe more issues need to be considered. The appearance in time is also of interest. Generally, if an error recurs rarely, but shows a regular appearance in time, it is of higher interest. A rare error that covers only a small amount of time from the total period, might be an isolated incidence. The reasons might be a normal configuration, hardware exchange or reboot etc. The low frequent algorithm will apply a technique to constrain only to errors that represent the time period.

Low frequent error theory

A low frequent error can be defined as:

when selecting a set of errors limited by time, and there is a unique error event exceptionally rare compared to the total amount of events, and where all other members in this set are repeatedly iterated commonly, we call this error low frequent.

Therefore it is a subjective judgment to define what is low frequent or not. In practice an algorithm given constant or input variable will decide where to set the limit in this matter.

An introduction of extended restrictions is desirable if we want to concentrate on low frequent errors that have a regular behavior. Again we apply set theory to formalize the algorithm requirements. First, let P represent a desired observation time period as:

$$P = \{t | t_{start} \leq t \leq t_{stop}\}$$

where the *start* and *stop* time in practice are operator chosen limits. Let e represent an error and let $T(e)$ be an error time function for an error event.

We then have a selected set S of errors where:

$$S = \{e | T(e) \in P\}$$

Further, let $F(x, y)$ represent a function counting the number of error repetitions for an error attributes **error** and **host**. Let $E(e)$ represent an element **error** attribut and let A represent a specific error attribute. Let $R(e)$ represent an element **host** attribute and let H represent a specific host attribute, c is a low frequent definition constant. Define the set

$$F(A, H) = \{e | e \in S \wedge E(e) = A \wedge R(e) = H\}$$

which is a function of A and H. Then the set F(A, H) represents a low frequent error if

$$|F(A, H)| < |S| \times c \quad (0 < c < 1).$$

The final selection of errors will have to pass two tests. These are:

- Time period representation.
- Regularity in pattern.

The time period representation is straight forward. The first error and the last must cover a certain time period out of the total selected period. The found errors must represent the selected time period to a certain extent, as discussed in the previous section. Again this is a subjective call and is decided from an algorithm constant. Let p be the constant for period representation, $\{t_1, t_2, t_3, \cdots, t_n\}$ be low frequent error event times where $n = C(e)$, and $\{t_{start}, t_{stop}\}$ be start and stop time of the investigated period. If the following mathematical condition for an error holds:

$$(t_{stop} - t_{start}) \times p < (t_n - t_1) \quad (0 < p < 1) \;\; (n > 1) \tag{2.1}$$

the time period representation is valid. An example is shown in figure 2.7. In the figure we also have $\{x_1, x_2, x_3, \cdots, x_{n-1}\}$ as time difference between events. These time values will come into calculation for an error regularity check. First we determine the ideal regularity λ from calculating:

$$\lambda = \frac{t_n - t_1}{n - 1} \qquad (n > 1) \tag{2.2}$$

The symbol λ now holds the value for a perfect regularity, that is, the average time between error events. Now we can summarize the differences between λ and $\{x_1, x_2, x_3, \cdots, x_{n-1}\}$ to see how much the pattern differs from the ideal regularity that has exactly the same time (λ) between all events. Since $x_1 = t_2 - t_1$, $x_2 = t_3 - t_2$ and so forth we can summarize

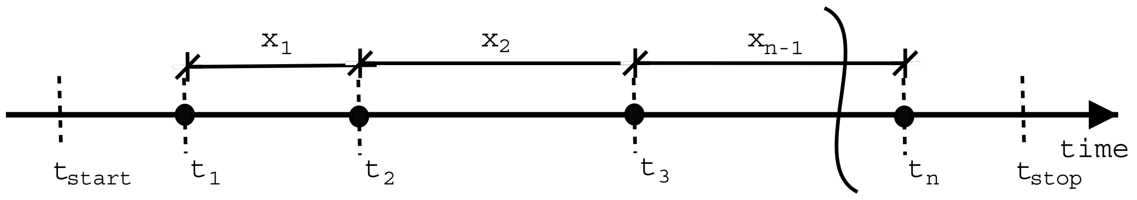


Figure 2.7: Example of low frequent error irregularity.
The dots represent the appearance in time of one low frequent error.

the differences as:

$$s = \sum_{i=1}^{n-1} |(t_{i+1} - t_i) - \lambda| \tag{2.3}$$

We now have the total sum of irregularity available. All there is left to do, is to check if the sum s, is small enough referring to what is desired by definition. The limit of maximum irregularity is controlled by a constant, named r. For the regularity validation, we also take into consideration the length of the error time period involved. Hence, the maximum deviation allowed will be proportional to the error presence in time.

This last calculation will eventually decide the low frequent error validity. If the following statement holds:

$$s \leq (t_n - t_1) \times r \qquad (0 < r < 1) \quad (n > 1) \tag{2.4}$$

the error behavior is of regular nature.

```
+-------------------+----------+------------+------------------------+
| rname             | ltime    | localdate  | errorcode              |
+-------------------+----------+------------+------------------------+
| kthnoc-2.sunet.se | 09:31:24 | 2003-08-28 | %GRPGE-6-GBIC_TX_FAULT |
+-------------------+----------+------------+------------------------+
1 row in set (8.35 sec)

+-------------------+----------+------------+------------------------+
| rname             | ltime    | localdate  | errorcode              |
+-------------------+----------+------------+------------------------+
| kthnoc-2.sunet.se | 10:22:38 | 2003-09-22 | %GRPGE-6-GBIC_TX_FAULT |
+-------------------+----------+------------+------------------------+
1 row in set (9.20 sec)
```

Figure 2.8: Output from *mysql* database.
Searching for error GRPGE-6-GBIC_TX_FAULT in Aug/Sep - 03. Tables have 933059 and 833936 rows respectively.

As shown, the theory covers the requirements for low frequent error isolation based on error behavior and appearance in time. The theory leaves room for a lot of experimenting with constants but also for modifications to the tests for regularity and representation in time. Only testing in a real environment will tell how well the theory adapts in practice and if modifications will be necessary.

Also, it is important to realize that the theory has the implication that, if only one event is found in a search period, it is disregarded. It might be of interest to consider this special case as a one time incident.

Case study

For a case study a database search for a known low frequent error in August and September is shown in figure 2.8.

This represents an interesting case, even though this particular error as such is uninteresting. It fulfills the criteria as it is rare, regular and is present a relatively large time of the total period, twenty five days out of sixty one. Also, it has the least possible events (two) to suite the theory which also has implications for the regularity control.

We make a brief summary of how these two events in figure 2.8 will be handled and if they can indicate a suitable value for the constant p in eq 2.1.

The conclusion is that this special case will work satisfying with the low frequent theory. Comparisons with other cases are hard to produce since low frequent errors are to hard to find. Hence, from this isolated event we can not make any qualified assumptions for a suitable constant value for time period representation in eq 2.1 and none for regularity control in eq 2.4. Regularity test has no meaning with only two errors involved. This confirms from:

$$\lambda = t_2 - t_1$$

for all cases in the low frequent theory as the sum from eq 2.3 will be zero and consequently eq 2.4 is true for all cases. We can only state that the constant p set to $p = 0.4$ will approve this case since by insertion into eq 2.1 we have:

$$61 \times 0.4 < 25$$

and therefore is a valid time period representation.

Algorithm issues

We have presented our theory of how the low frequent calculations depend on constants. These constants will be provided in the algorithm implementation as definitions after testing. But some extra algorithm variables could be necessary to provide as program arguments for better adaption to the environment, or for the situation at hand. These are:

- Maximum number of events to regard an error as low frequent.

  - The algorithm must decide an upper limit for the number of events allowed for an error to qualify as low frequent. In some cases this number might be too low and should easily be possible to increase when needed.

- Max grade of error severity to process.

  - The operator might want to disregard errors of low severity and should be allowed to set the level as a program input argument.

Searching for low frequent errors are presumed to take place over a considerably long time period and will therefore need to handle and process a large amount of data. Hence, efficiency will be important at implementation time and memory consumption as well as CPU usage might be limiting factors for longtime period processing.

2.3.4 Network log deviation

On a single point of failure a network component might suddenly generate more data. It might be an isolated effect or spread to neighbouring parts. Doing regular measurements on the log generation intensity could reveal if an abnormality occurs. This task would only be meaningful if intensity checks are performed on a regular basis in small intervals. If a deviation is found a report could be sent to the network surveillance staff.

Log deviation theory

To register a deviation we first need some reference for comparison. On a regular basis a value for normal generation could be carried out in the present month. Suppose we are at the tenth day in a month and we have at present 250000 messages in this month we will have an approximate average log rate a as:

$$a = \frac{250000}{(10 \times 24 \times 60)} \approx 17.36 \;\; logs/min$$

If a comparison of this value is done with the last minute average log generation, repeatedly every minute, action can be taken if we reach a predefined critical limit. We can also adjust the average value dynamically every new day, and start all over again in a new month, to produce an average that adjusts better to the network behavior.

Since deviations can be of different size, different levels of reports ought to be generated. Let c represent the last minutes' average of log generation intensity. Then the following example could be a possible differentiation and actions performed if:

$$c < 2a$$

No action required.

or if:

$$2a \leq c < 4a$$

Send: Warning.

or if:

$$4a \leq c < 8a$$

Send: High intensity state.

or if:

$$8a \leq c$$

Send: Critical state.

These boundaries have to be determined by experience.

If a send level is triggered by the algorithm that generates the warning, it also has the possibility to count the number of logs generated for all network components involved.

This data could be attached to a warning in descending order to provide a quick overview of the situation.

Log deviation algorithm issues

It is necessary to realize that this part of extracting information from the log data approach real time surveillance. It is possible to realize, but has a questionable purpose since other tools are available using SNMP technology as a base. To use an analogy, there is no point in reinventing the wheel in an other material.

We have shown that in theory, log deviation detection is possible, but has a questionable purpose when using network router log data. If information about traffic is needed SNMP is commonly used. Technicians only need to log in on a router and check router traffic variables. They can also use a SNMP based tool that retrieves the wanted data from a router.

Investigation in log deviation has shown that the increase of log data are most often generated upon configuration changes in the network. These changes are made by technicians at the NOC[9] and are known to increase log data. A log deviation algorithm would give information already known to the NOC. Because of these facts log deviation will not be considered any further in this dissertation.

[9]Network Operation Centre

2.3.5 Link failure detection

It is not difficult to detect when the network is down for some reason, but if some part of or the entire network is only down for a few seconds nobody might notice. One additional reason that contributes to this fact is that the network architecture often is built redundant. That is, if one link fails, the traffic immediately takes another path in the network to reach its destination. But still if one link fails, it is considered to be a serious problem. As mentioned in the introduction chapter, there are often financial aspects involved with link failures.

There are two approaches towards link failures. One is to generate immediate alerts on events and the other is to perform analysis on historical data. Detecting link failures can be done by network monitoring tools that monitor the events in a network, but this requires additional personnel at the NOC monitoring in realtime. The goal here is to make this analysis possible from the historical log data to confirm cases and reveal unknown link failures.

We will discuss the possibilities for creating a state machine of the core network's state. It is not only the link itself that could be broken, the interface managing the link could also be down and must also be considered. To be able to draw any conclusions, full knowledge of the complete network connectivity is essential.

Network link connectivity information

To analyse the network link status the network connectivity has to be studied and understood. If a link fails and an error message is generated, this has to be confirmed from the router at the other end of the link. Hence, we must be certain of which additional router must be involved.

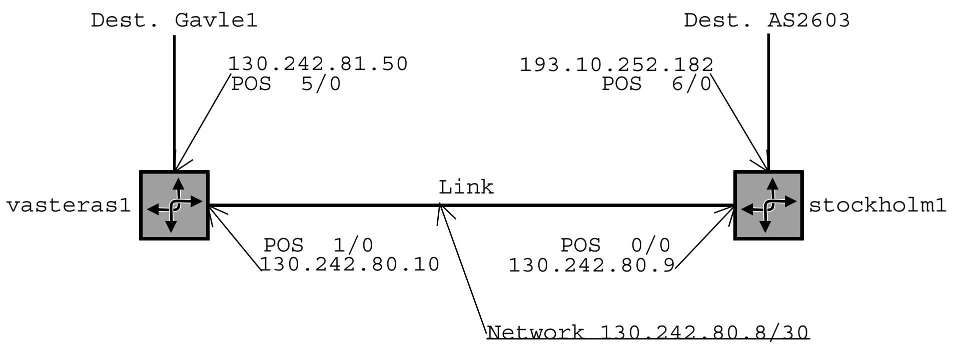


Figure 2.9: Schematic view of connectivity from part of network.

Showing network 130.242.80.8/30 with netmask 255.255.255.252 defining one specific link. This link physically joins routers vasteras1 and stockholm1 to exchange data on POS interfaces.

All routers in the network have several interfaces each belonging to some network with a unique IP-address. Hence, each interface that connects via a link to some other router, belongs to the same logical network as the interface on the opposite side. This is the key to find out how the routers are physically connected. This is an inevitable task for analyzing link status since the interfaces at both sides have to be studied simultaneously to be able to make correct conclusions. This matter will be discussed further after studying the network connectivity principles.

In figure 2.9 two routers are shown and how they are connected with some other surrounding neighbours by POS[10] interfaces. If we study the figure we can see that interface POS 1/0 in *vasteras1* is connected with POS 0/0 in *stockholm1*. From only reading the interface IP-addresses and netmask, we can draw this conclusion by noticing that they belong to the same defined logical network, namely 130.242.80.8/30.

Before starting to analyse the link status in the network, the required data for all in-

[10]Packet Over Sonet. The connection is similar to a serial Point-To-Point connection. Other standards exists, such as SDH - Synchronous Data Hierarchy, commonly used in router technology in European core networks. The additional information after POS, 0/1 gives information on slot number in router and index on interface respectively.

Router	**Interface**	**IP**	**Netmask**	**Network**
vasteras1	POS 1/0	130.242.80.10	30	130.242.80.8
vasteras1	POS 5/0	130.242.81.50	30	130.242.81.48
stockholm1	POS 0/0	130.242.80.9	30	130.242.80.8
stockholm1	POS 6/0	193.10.252.182	30	193.10.252.180

Table 2.1: Table showing a fictitious database table with necessary data for knowledge about network link connectivity.

Here [vasteras1 POS 1/0] and [stockholm1 POS 0/0] represent one link since they belong to the same logical network.

terfaces in the network routers has to be collected and made available for an analyzing algorithm to read. This can be done with SNMP[11] to get raw data for insertion into a database table by some script or program. And by applying this to all involved routers in the network, the connectivity can logically be determined for all existing links. A fictitious table from collecting information with SNMP is shown in table 2.1 representing the network part in figure 2.9.

If some interface indicates a link failure, we can with help from this table draw further conclusions about true physical link malfunctioning or disregard the event as another problem. The reasons for a failure could also origin from an interface hardware or software error.

For example, if *vasteras1 POS 1/0* changes state to down and so does *stockholm1 POS 0/0*, from studying table 2.1 we know a link between those routers might experience some sort of malfunctioning. It could simply have been cut off by accident.

[11]To extract this data with SNMP one can read router variables as: [snmpwalk -v 1 -c public *routername* ifDescr] to get a list with all interfaces and [snmptable -v 1 -c public *routername* ipAddrTable] to get a table with IP and netmask for all interfaces.

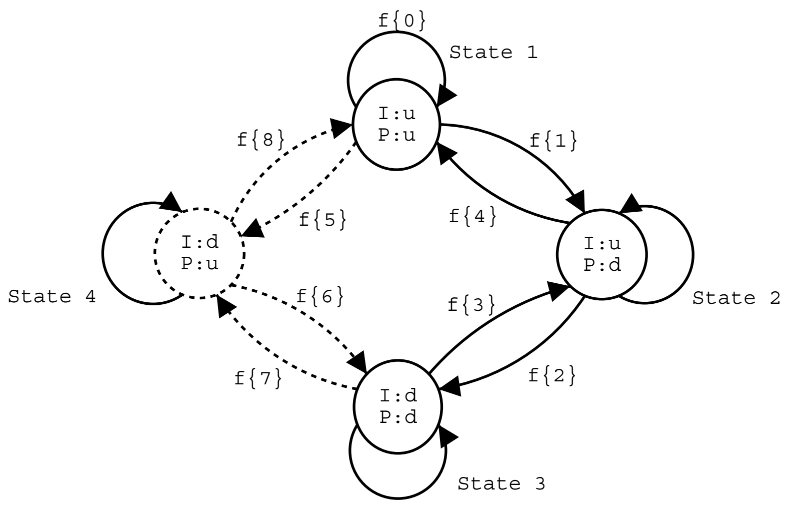


Figure 2.10: Router interface *Link State Machine*.

The *LSM* shows possible changes of state for interface and line protocol up/down (u/d) at link failure detection. (I) is router interface and (P) line protocol. Dotted lines represent changes caused by other failures than link related.

Link failure investigation algorithm theory

To investigate if a link failure has occurred, it is suitable to define a state machine for each interface involved during an incident. Such a machine is presented in figure 2.10. There are two objects involved with link failures, the link interface and the line protocol belonging to the interface. Each of these can have two different states, up or down, but the sequence of changed states for a link failure are restricted because of technical reasons. E.g., the line protocol can not change state from up to down until the interface does so, and vice versa. Else some other internal router problems probably exist irrelevant for link analyzing, e.g, a broken interface is most likely to cause a link problem.

The analyzing algorithm needs to set up a state machine for both interfaces involved during a link failure to evaluate link status by analyzing both machines concurrently. If one interface changes state, the algorithm detects the corresponding interface at the other side

```
NR  ROUTER NAME         ERROR               MESSAGE DATA                                             TIME

1)  stockholm1.sunet.se %LINK-3-UPDOWN      Interface POS0/0,                  changed state to down 17:01:19
2)  stockholm1.sunet.se %LINEPROTO-5-UPDOWN Line protocol on Interface POS0/0, changed state to down 17:01:22
3)  stockholm1.sunet.se %LINK-3-UPDOWN      Interface POS0/0,                  changed state to up   17:03:00
4)  stockholm1.sunet.se %LINEPROTO-5-UPDOWN Line protocol on Interface POS0/0, changed state to up   17:03:00
5)  vasteras1.sunet.se  %LINK-3-UPDOWN      Interface POS0/0,                  changed state to down 17:03:16
6)  vasteras1.sunet.se  %LINEPROTO-5-UPDOWN Line protocol on Interface POS0/0, changed state to down 17:03:18
7)  vasteras1.sunet.se  %LINK-3-UPDOWN      Interface POS0/0,                  changed state to up   17:04:45
8)  vasteras1.sunet.se  %LINEPROTO-5-UPDOWN Line protocol on Interface POS0/0, changed state to up   17:04:50
9)  vasteras1.sunet.se  %LINK-3-UPDOWN      Interface POS1/0,                  changed state to down 17:09:29
10) vasteras1.sunet.se  %LINEPROTO-5-UPDOWN Line protocol on Interface POS1/0, changed state to down 17:09:30
11) vasteras1.sunet.se  %LINK-3-UPDOWN      Interface POS1/0,                  changed state to up   17:10:48
12) vasteras1.sunet.se  %LINEPROTO-5-UPDOWN Line protocol on Interface POS1/0, changed state to up   17:10:48
```

Figure 2.11: Output from log database.
The output presents some link failure scenarios from 2003-07-17 with routers stockholm1 and vasteras1.

of the link as in the earlier theory for network connectivity. To complement the analysis in figure 2.10, a table of states need to be consulted with predefined possible states. Note that one state is considered to be invalid for link failures in figure 2.10 and is therefore disregarded in this matter as not being a physical link related problem. In theory, only functions $\mathbf{f(1)}$ to $\mathbf{f(4)}$ can therefore be involved. If some other movement occurs, one could disregard this as being irrelevant regarding a physical link failure.

A table of states is shown in table 2.2. Some combined states are considered to be steady. That is, those states are possible static states with a broken link or with a flawless link.

To exemplify the use of the table with an algorithm, a snapshot from link associated log data has been collected from the log database. This data is shown in figure 2.11 where all data is filtered[12] to show only link relevant data from 2003-07-17 from 17:00 to 17:15.

An example of how the algorithm is meant to work with this data begins when row one is found. First the the algorithm search for *stockholm1 POS 0/0* in table 2.1 and the interface belonging to the same network, *vasteras1 POS 1/0*. For those two interfaces two

[12]The database query was set to match errors beginning with ”LINK” or ”LINEPROTO”.

Steady State	**Router 1**	**Router 2**	**Link failure**
Yes	I:u P:u (S1)	I:u P:u (S1)	No
No	I:u P:u (S1)	I:u P:d (S2)	Possible
No	I:u P:u (S1)	I:d P:d (S3)	Possible
No	I:u P:u (S1)	I:d P:u (S4)	Other failure
No	I:u P:d (S2)	I:u P:u (S1)	Possible
Yes	I:u P:d (S2)	I:u P:d (S2)	Likely
Yes	I:u P:d (S2)	I:d P:d (S3)	Possible
No	I:u P:d (S2)	I:d P:u (S4)	Other failure
No	I:d P:d (S3)	I:u P:u (S1)	Possible
Yes	I:d P:d (S3)	I:u P:d (S2)	Possible
Yes	I:d P:d (S3)	I:d P:d (S3)	Possible
No	I:d P:d (S3)	I:d P:u (S4)	Other failure
No	I:d P:u (S4)	I:u P:u (S1)	Other failure
No	I:d P:u (S4)	I:u P:d (S2)	Other failure
No	I:d P:u (S4)	I:d P:d (S3)	Other failure
No	I:d P:u (S4)	I:d P:u (S4)	Other failure

Table 2.2: Table showing the probability for link failures for state machines of router interfaces and their common physical link.

(I:) is interface up/down (u/d) and (P:) is line protocol up/down respectively. **No** indicates no link failure, that is, the state presumed normal. **Other failure** indicates some other reasons for link related problems such as hardware interface faults etc. **Possible** is indicating a state that could be a prestate to an upcoming failure or when leaving a true link failure. A steady state **Yes** indicate a static state in which both state machines could stay until external conditions change. All states that are not **steady** are considered as a transitions phase of some sort. They should normally only exist for a short period of time while moving to some other steady state while considering link status.

Steady	**Event**	**SM vasteras1**	**SM stockholm1**	**f()**	**Time**	**Link failure**
Yes	-	I:u P:u	I:u P:u	0	**00:00:00**	
No	1	I:u P:u	I:d P:u	5	17:01:19	Other
No	2	I:u P:u	I:d P:d	6	17:01:22	Possible
No	3	I:u P:u	I:u P:d	3	17:03:00	Other
Yes	4	I:u P:u	I:u P:u	4	17:03:00	No
No	5	I:d P:u	I:u P:u	5	17:03:16	Other
No	6	I:d P:d	I:u P:u	6	17:03:18	Possible
No	7	I:u P:d	I:u P:u	3	17:04:45	Possible
Yes	8	I:u P:u	I:u P:u	4	17:04:50	No
No	9	I:d P:u	I:u P:u	5	17:09:29	Other
No	10	I:d P:d	I:u P:u	6	17:09:30	Possible
No	11	I:u P:d	I:u P:u	3	17:09:48	Possible
Yes	12	I:u P:u	I:u P:u	4	17:09:48	No

Table 2.3: Table showing the state machine events for figure 2.11.

Link failure probability are set from consulting table 2.2. The function f() represents a movement in the state machine shown in figure 2.10. All steady states are marked with *Yes*. The transaction scheme in this table does not revel any steady state with *possible* or a *likely* link failure.

state machines are set up as in figure 2.10. The state machine starts initiated as being in **State 1** presuming the link was up and running. We present the result in a new table 2.3 to show how state events from log data in figure 2.11 are evaluated in a step by step procedure.

For all changes of state, table 2.2 is consulted for possible link failure. This data did not indicate any probability for a link failure. The state transaction functions **f(5)** and **f(6)** also indicates for *other failure* involved. This happens because the interface changes to state down before the line protocol does, and this is not consistent with what to expect from a link failure. However, this inappropriate order is at present often the case due to an internal router bug. This means that transition **f(5)** and **f(6)** in figure 2.10 also should be accepted as normal, if the transaction **f(6)** occurs within approximately five seconds after **f(5)**. Hence, link failures in table 2.2 need to be updated to accept **State 4** [I:d P:u] as *possible* to adapt to this internal router fault, if implemented at present situation.

However, adapting to this bug or not is not critical since the **State 4** is an unsteady state for all cases involved. If a link failure should be *likely*, one must arrive at some steady state while considering both link state machines, something that is expected to happen if the link is physically disconnected or broken.

The time period being in a steady state is considered to be the period for the link failure. In a similar way for a *possible* failure, but with less fault probability. The time period together with the probability factor *possible* or *likely* will serve as algorithm variables to investigate for a report generation.

Link failure summary

The task of detecting link failures is not trivial. Several scenarios are possible and one must also take into consideration that logs might be lost during link failures. Something that complicates reliable reporting. If an analysis finds a possible link failure, it will always have to be validated by a network technician in all cases. Other incidents could be summarized and reported as interface problems, if wanted. Still, the conclusions from the investigations and the creation of a theory show possible usefulness. Even if limiting factors are present and algorithm accuracy shows would not be better than about fifty percent, it could still be worthwhile to realize.

A part of this thesis has been to investigate the possibility of link failure detection and create a foundation for implementation. This has proved to be possible and shows potential from making further use of the historical log data kept in database. Further discussion and conclusions will be presented in *future work*, chapter 5.3.

2.4 Summary

The project outline ideas have been presented and theories discussed to extract more information from the network historical log data. If relations in this data can be found and reported, it can assist the daily network troubleshooting. The hypothesis is to do so by revealing unknown error states. Some does not instantly cause network disturbance, but might in the future, if overlooked and not attended to.

Implementation details will follow in the next chapter where selected theories will be implemented in task specific algorithms that analyse the data and produce reports.

Chapter 3

Experiment and Implementation

3.1 Implementation objectives

In this chapter algorithms from the issues described in the background chapter will be presented. The goal is to provide information of interest by analysing historical events. Implementations made should be easy to use and should require a minimum of human surveillance. Applications written are intended to be run as jobs on a monthly or weekly basis. They are not surveillance software. These software parts are purely computations and will not require user input after starting the respective process. No real time analyzing of the incoming data will be considered.

The design goal has been to cause as little workload as possible on the database since it also is constantly managing new data and provides other important services. Due to this it is important to avoid work overload on the database. Preferably, the algorithm process should not run on the database host for the same reason.

The research is divided into several subareas to provide possibilities to evaluate the result from each subarea. The main objectives are areas which can be tested and evaluated by empirical observations. We know log rows might relate in a chain of events or originate from a single point of failure. The research will focus on finding relations between log entries in the database.

3.2 Network environment

The algorithms implemented as programs are intended to work in a networked environment thus not running on the database host. The programs can be used without a network connection and run on the same machine as the database host but this is not recommended as the programs make demanding computation. All programs will connect to the database once and retrieve data and then disconnect for data processing. A common configuration file with authentication information for the database connection will be used by all programs. It is in fact the same file as used in the base system, see [1] for further information.

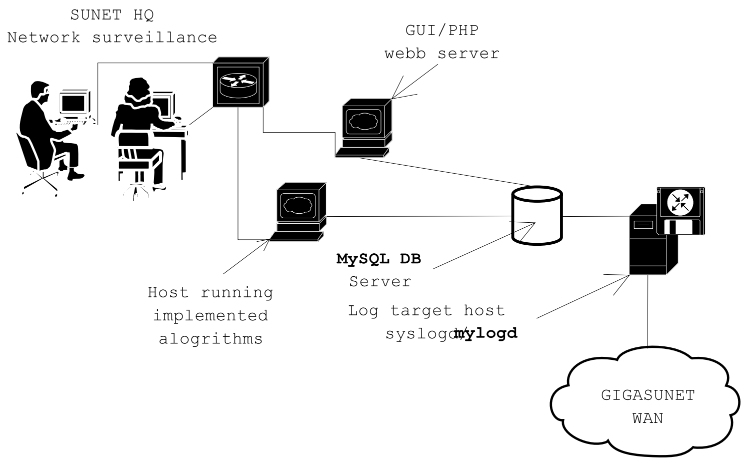


Figure 3.1: Base system schematic overview with new component

Figure 3.1 depicts the environment the developed algorithms intend to run in. The outcome of the algorithmic processing will be evaluated for its significance regarding the network fault detection area in chapter 4. Diagnosing the failures will not be considered and is up to the network administrator.

3.3 Implemented algorithms

The implemented algorithms from theory are presented here. They represent one possible implementation and are all first of all meant as tools to test selected theories from chapter 2.1.

3.3.1 Diffusion error detection

Error diffusion has been chosen based on experience by KTHNOC as a possible critical failure that might go undetected. Evaluation of error diffusion is critical to prevent future repetition. Automated diffusion detection is the goal for the algorithm in this area, as discussed in section 2.3.1.

In section 2.3.1 error diffusion was defined as:

One or more errors that within a limited period of time are reported from two or more network components

That is, an initiated error will spread from one router to neighbouring routers causing them to report the same error in a chain reaction. Errors observed so far tend to disappear within only a few seconds, this makes them hard to find by manual inspection.

Figure 3.2 shows how an assumed error diffusion in the network might look. It occurs when a specific error, in a short period of time, suddenly is present in several routers. Before defining an algorithm for diffusion two parameters must be defined:

- The minimal number of routers involved to be evaluated as diffusion
- The time interval for error evaluation.

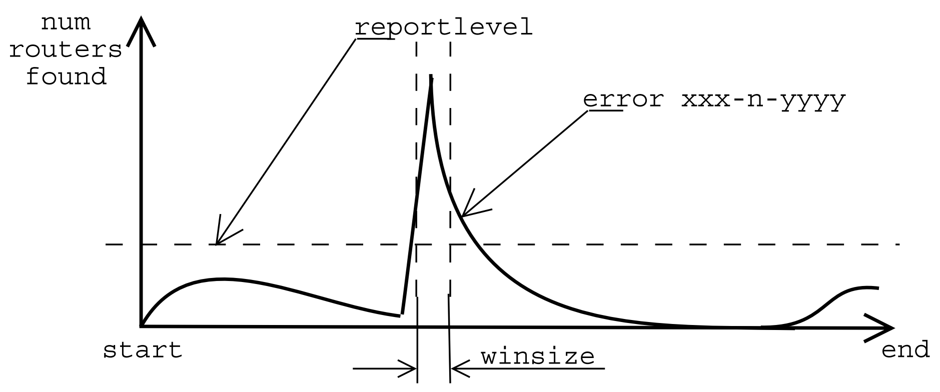


Figure 3.2: Assumed error diffusion in network
reportlevel: The minimal number of routers that have the same error.
winsize: The time slot to search for diffusion.
error xxx-n-yyyy: A specific error

In figure 3.2 the minimal number of routers involved to be evaluated as diffusion is represented by *reportlevel* and the time interval for error evaluation as *winsize*. There is a reason for this name which will be explained later. These parameters have to be set by the operator. The values are estimated and have to be adjusted from experience with an working algorithm.

To verify the impact of network diffusion errors, real data was manually collected from a typical incident. The diffusion starts at 12:45:03 and ends at 12:45:09. Observing a period from 14:40:00 to 14:50:00 will capture some of the normal network state. A plotted graph from the event is shown in figure 3.3.

At this diffusion there was an error spread, BGP-6-NLRI_MISMATCH. The plotted graph shows how the error, in a period of ten seconds, is present in fortyfive different routers. The normal state of network does not give more than three errors of the same type within a ten second period.

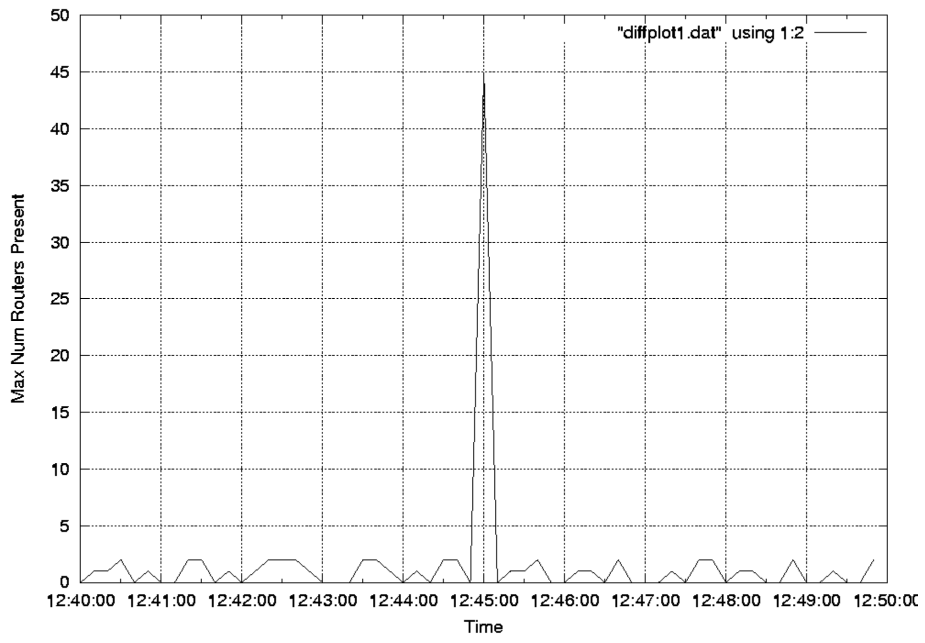


Figure 3.3: Error diffusion plot 2003-02-26, 14:40:00 - 14:50:00
Generated with time-window set to 10 seconds.
Error diffusion cause, showing BGP-6-NLRI_MISMATCH.

Searching manually the database February table reveals a total amount of three diffusion errors with no more than twenty seconds of diffusion state in total. This makes them hard to detect if they don't cause any side effects that will call the network technicians' attention.

To provide automated detection and reporting, an algorithm will be needed. The goal is to have a flexible and efficient search with small workload on the database. The algorithm will search for diffusion by moving a *time-window* over the data to evaluate.

Algorithm description

The algorithm will find error diffusions by looking at errors in a small time period. Errors are collected from the database based on a selected time interval, maximal length of one month. The list containing retrieved errors will be investigated in small chunks at a time. These chunks represents a time period of logged errors and are defined as *time-windows*. In these *time-windows* errors will be counted and evaluated.

The number of routers that have the same error type in a *time-window* are counted and compared to the parameter *reportlevel*. The *reportlevel* parameter determines if the sequence of error is a possible error diffusion. If the number of errors is greater or equal to *reportlevel* further processing is needed. The other conceivable case with less than *reportlevel* will be described later. Figure 3.4 illustrates a *time-window* with the list of errors placed on an axle. Every error in the *time-window* must be checked for its type and its originator. The type is used to distinct the error, BGP-6-NLRI_MISMATCH and BGP-5-ADJCHANGE are examples of two different types of error. An error must have at least *reportlevel* originators to match as a diffusion.

For example, if five errors appear in the time-window and the *reportlevel* is set to five, the type of the errors and their originator must be checked. We define *DiffList* as: A list of originators for an error, the originators should all be unique. That is, the network components that produced the same error is the nodes in *DiffList*. The size of *DiffList* (the number of network components/list nodes) must be greater than or equal to *reportlevel* to be a diffusion.

In table 3.1 there are three different scenarios, for some time-window. In the columns the originators are identified as *router A, router B* and so on. In the leftmost column there are three distinct routers and hence a diffusion, if *reportlevel* is three. The middle column has only two routers so error Y is not considered as a diffusion. In the rightmost column

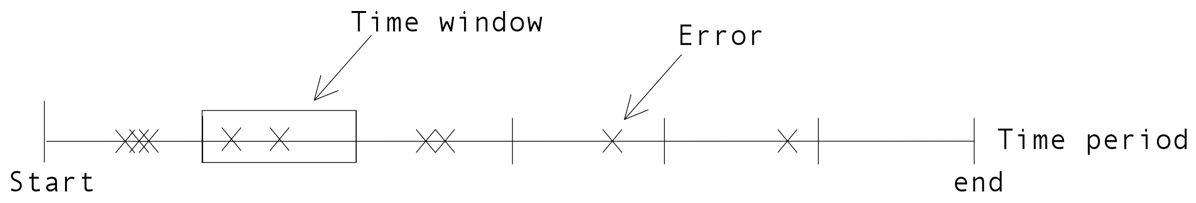


Figure 3.4: *time-window*
Within the *time-window* errors are counted for further processing.

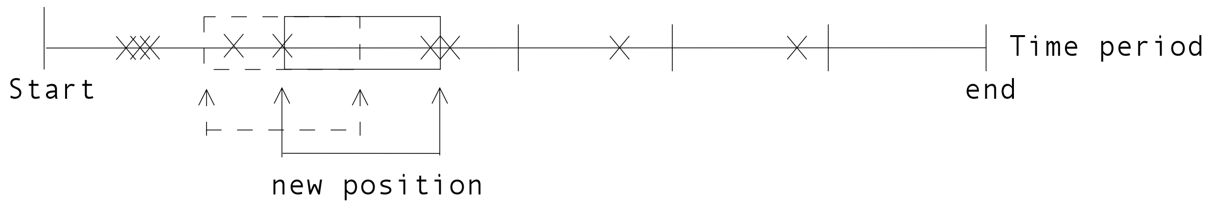


Figure 3.5: New position of the *time-window*
All errors in the time-window have been processed and the time-window is moved forward half its size in seconds.

there are three routers but one router appears twice, so one *router A* will be discarded and only two originators would be place in *error Z's DiffList*.

Error **X**	Error **Y**	Error **Z**
router A	router A	router A
router B	router B	router A
router C		router D

Table 3.1: Diffusion
Example of processed data in one *time-window*, *reportlevel* at 3

In figure 3.6 the *time-window* is shown enlarged. The routers are named by a number and the error type by a letter. In this figure a window with five errors has been selected. The error **A** is occuring in the routers **1, 3, 5** and **7**. If now *reportlevel* is at most four this will be caught as a diffusion.

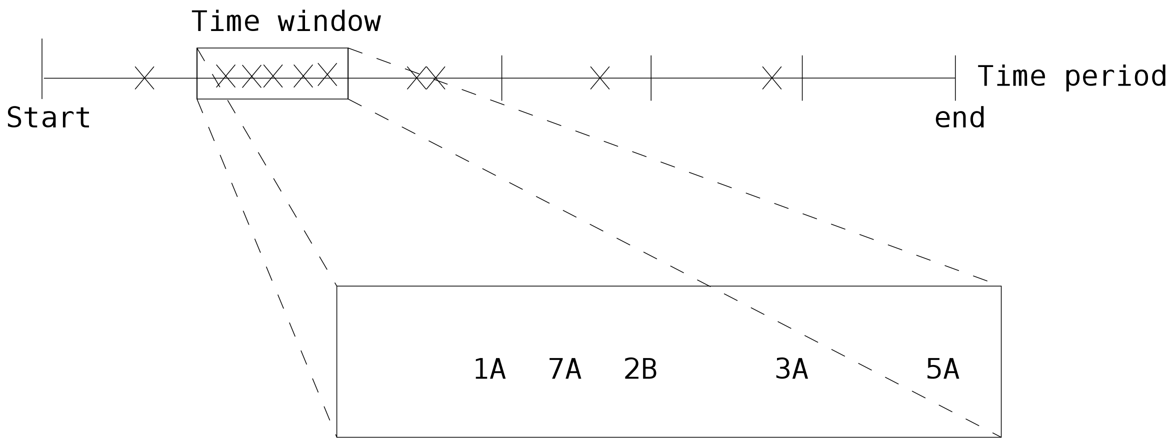


Figure 3.6: Zoomed time-window.

If the number of errors is less than *reportlevel* the *time-window* must be moved, (investigate a new chunk of data from the error list). The time-window will move in steps of half its size to detect any diffusion that starts in one window and ends in the next. Figure 3.4 and 3.5 give a view over the time-window and its movement. One problem that can arise of this is that the same diffusion can be detected twice. If the time-window is from zero to ten seconds and a diffusion is detected between the five and ten second interval, the next position of the time-window between five seconds to fifteen will also detect this diffusion. This scenario is shown in figure 3.7. Figure 3.8 gives a view of overlapping time-windows and why it is used. In the implementation the latter time-window is discarded in these cases. Note, if a diffusion extends the time-window it will be missed.

A scenario that could happen with a time-window that move forward more than one second is that diffusions can be missed. An example of this is shown in figure 3.9. To detect all diffusions the time-window must be moved forward one second since the time of all errors are stored in seconds.

The algorithm will only cause one operation on the database leaving it idle for other work. This operation involves retrieving data for the selected time interval. The algorithm

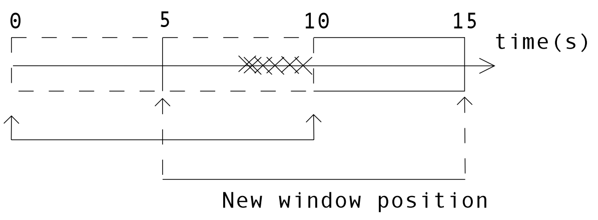


Figure 3.7: The same diffusion detected twice

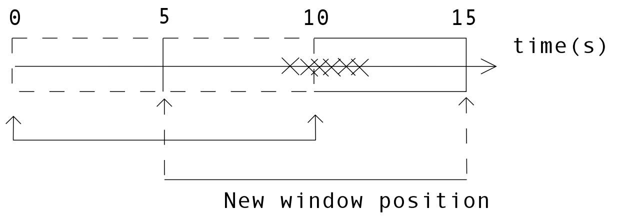


Figure 3.8: Diffusion overlapping two time-windows
The time-window moves in steps of half its size to detect any diffusion that starts in one window and ends in the next.

described in this chapter can be studied in appendix A.

3.3.2 Recurring sequences

In this section an algorithm description for recurring sequences will be presented. The algorithm will be based on the information in chapter 2.3.2. Details about the algorithm

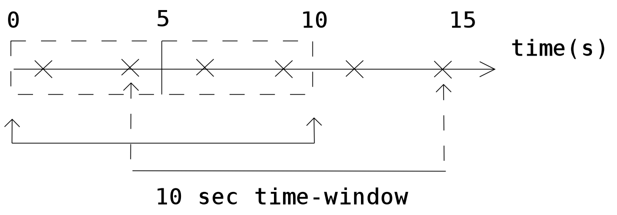


Figure 3.9: A missed diffusion, with a reportlevel set to 5
The 10 second time-window will miss the diffusion between 4 and 14.

will be given in figures with describing text.

If patterns exist, are the errors in the pattern related to each other? Do they cause each other? The number of repetitions and length of a sequence should indicate this and rule out the possibility that the pattern was created haphazardly. The main issue in this chapter is to develop an algorithm that finds patterns in the archived error logs. A pattern was defined in chapter 2.3.2 as a repetition of a sequence of routers that are repeatedly reporting the same error in that sequence. An example of this was shown in figure 2.5

Algorithm description

An important function of the algorithm is to point out the starting router for the sequence. Common troubleshooting involves locating the problem cause and taking care of it from there. Cases where the origin for a sequence is doubtful are not managed by the algorithm but will be discussed in this chapter.

Definitions in recurring sequences considered in chapter 2.3.2:

- The number of repetitions of the sequence, *seqrep*
- The minimum length of the sequence, *seqlen*

The item "how to avoid reporting a subsequence of a sequence" discussed in chapter 2.3.2 will be discussed in the *detailed algorithm information* section.

For a sequence to be a valid recurring sequence (pattern) these conditions must be met:

- The length of the sequence must be at least of length *seqlen*, which is user specified and would reasonably be at least 10% of the number of routers in the network. This is very dependent on the type of network and its size. Having *seqlen* less than 10% of the total number of routers in GigaSunets would give a tremendous amount of patterns and not be of much use in troubleshooting. These patterns which are pointed out in chapter 2.3.2 could have been created by mere accident.

- The number of repetitions for the sequence must be greater or equal to *seqrep*, which is also user specified. *Seqrep* should be more than 2 to rule out sequences created by mere accident. This parameter does not rule false sequences totally but minimises the risk.

The balance between *seqrep* and *seqlen* is up to the technicians responsible for troubleshooting the network to investigate further. First we define the number of repetitions for a sequence as:

$$\text{*sequence} = \text{Number of actual repetitions for a pattern}$$

and we summarize the conditions for a valid pattern in eq 3.1.

$$|sequence| >= seqlen \;\; AND \;\; (*sequence) >= seqrep \tag{3.1}$$

For recurring sequence three main scenarios have been put together to demonstrate which patterns and how they are managed, A, B and C. The three sequence cases will show how the algorithm works and the thoughts behind it. Routers are named by numbers and errors by a letter, e.g. 1A, 2A, 3B; router 1 and 2 have the same error. Circle indicates start.

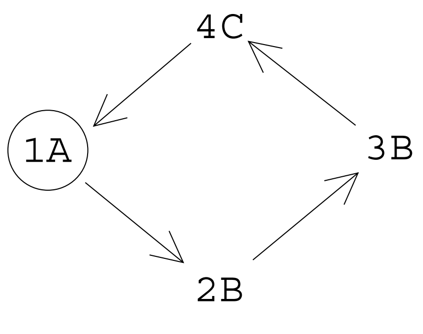


Figure 3.10: Case A: {1A, 2B, 3B, 4C, 1A} Not managed, Source not reliable.

The source, i.e. the starting node for the sequence in case A, figure 3.10, can not be set for sure based on its position. It gives an indication but not a reliable source for the initiating of the sequence. Based on this argument sequences like in case A is not managed by the algorithm.

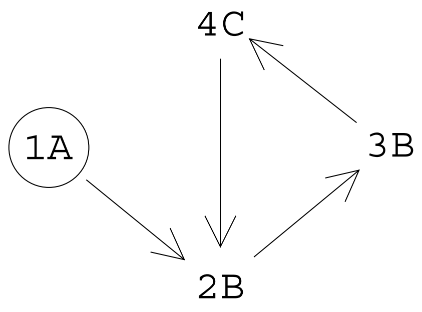


Figure 3.11: Case B: Managed if 1A is the starting node.

Case B, in figure 3.11, is similar to A but here the starting node is separate from the loop {2,3,4}. 1A must be the starting node for the sequence and the terms for a valid sequence are met.

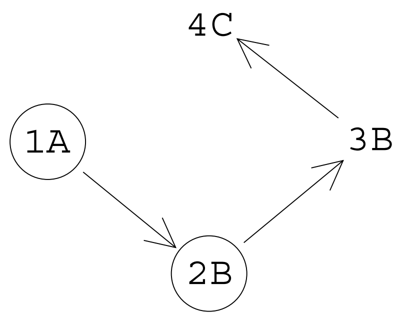


Figure 3.12: Case C: Managed, 1A and 2B can be starting nodes; *seqlen* 4 or 3.

In case C, shown in figure 3.12, either node can be the starting node because there are no loops in the sequence. The terms for a valid sequence must also be met. In figure 3.12

1A and 2B are marked as possible starting node with *seqlen* 4 or 3, *seqrep* is not considered here.

Detailed algorithm information

All data fetched from the database is put into a list for further processing. This list contains information such as time of the event, router identification (id) and the error code for the specific error. Based on the router id and error code the algorithm will search for recurring sequences in the list. The user given parameters *seqlen* and *seqrep* specify how many entries there must be in a sequence and how many times the sequence must exist in the list. The algorithm uses *seqlen* as the minimum length of a sequence. If a sequence of length *seqlen* has been found, the following entry in the list is also compared until no more match can be made. We define *maxlen* to be the maximal number of matched entries for a sequence.

This is demonstrated in figure 3.13 where *seqlen* is equal to three where the algorithm continues until the compared sequences do not match. In this case 5D is the last equal.

Sequences that begin with a loop, e.g. {1A 2B 3B 1A 4E}, will not be managed as discussed previously in the case examples.

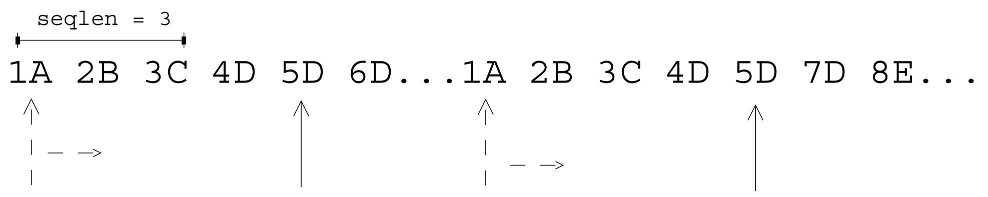


Figure 3.13: A recurring sequence, a pattern is found.

The focus lies on the triggering router for the sequence, avoiding subsequences gives less unwanted information and more focus on the triggering fault. Therefore when a matching sequence is found and no more sequences can be found, the algorithm skips forward *maxlen* steps to a new sequence. The reason for moving *maxlen* steps is that we do not want to search for subsequences. Since the algorithm finds and saves all matching sequences with $\geq$ *seqlen* no wanted sequences will be skipped with this procedure. For a sequence to be a pattern it must be found at least twice in the retrieved list, a recurring sequence.

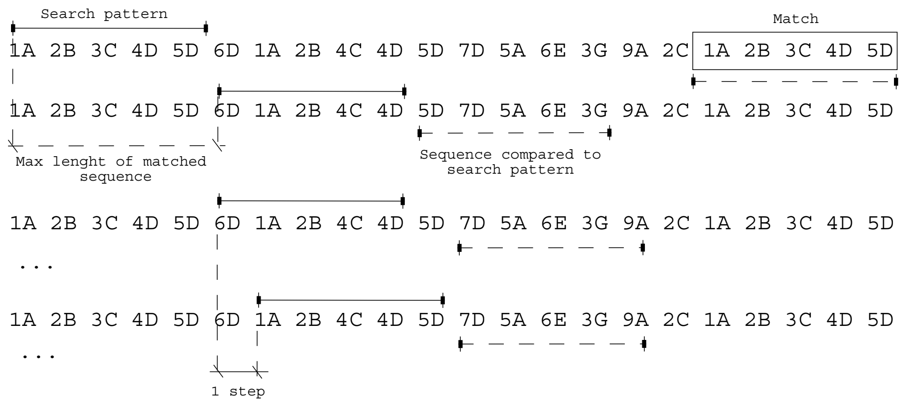

Search pattern
Match
1A 2B 3C 4D 5D 6D 1A 2B 4C 4D 5D 7D 5A 6E 3G 9A 2C 1A 2B 3C 4D 5D
1A 2B 3C 4D 5D 6D 1A 2B 4C 4D 5D 7D 5A 6E 3G 9A 2C 1A 2B 3C 4D 5D
Max lenght of matched sequence
Sequence compared to search pattern
1A 2B 3C 4D 5D 6D 1A 2B 4C 4D 5D 7D 5A 6E 3G 9A 2C 1A 2B 3C 4D 5D
...
1A 2B 3C 4D 5D 6D 1A 2B 4C 4D 5D 7D 5A 6E 3G 9A 2C 1A 2B 3C 4D 5D
...
1 step

Figure 3.14: List in which patterns are found.

Figure 3.14 shows a list of router id and error code, with *seqlen* $= 5$. This is the initial search, the sequence that will be tried to match other sequences will be {1A, 2B, 3C, 4D, 5D}. This sequence results in a match at the end of the list. The found sequence is saved for comparison and as a result, the algorithm will not search for the same pattern twice. When no more match can be found for the current pattern a new search sequence is set. Since the {1A, 2B, 3C, 4D, 5D} sequence was found the next sequence will begin at position directly after the current pattern, as shown in figure 3.14 at line two. If no match had been found the new search sequence had been set at the next position, i.e. 2B. The sequence that is compared with the search sequence is changed to its successor which is the next in the list, row three in the figure. When the compare sequence reaches the end of the list, the search pattern is changed as shown at row four in the figure.

If the search sequence should be a pattern already matched it will be skipped and the new sequence will start after the already found pattern. Figure 3.15; (1) shows this case.

After sequence {1A, 2B, 3C} has been matched and no more sequences can be found the new sequence is set according to **(1c)** in the figure. The procedure is that the new search sequence should be set *maxlen* steps after the searched sequence. The new sequence is {1A, 2B, 3C} which is shown in the figure, **(1b)**. But since this sequence already has been found as a pattern the new sequence will start at the position directly after the found pattern, as shown in **(1c)**.

(2) in figure 3.15 demonstrates case A with a not reliable source which was discussed earlier in this chapter. Instead of {1A, 2B, 3C, 4D, 5D, 6D, 1A} being matched only {1A, 2B, 3C, 4D, 5D, 6D} will be matched. When all sequences that match the {1A, 2B, 3C} sequence a new search sequence is set as shown in **(2b)**.

In **(3)** a demonstration is given if a search sequence contains a initiating loop. The algorithm will search for matching sequences but it will fail to get *seqlen* matches. When a loop is detected the sequence will be discarded, both in **(3a)** and **(3b)**. In **(3c)** the search sequence has moved past the initiating loops and in this case it can be matched.

As described the algorithm only focus on the router *id* and error, not the time when the error occurred. This could lead to sequences where the time between events in the sequence could question the events as related[1]. These sequences should, however, be eliminated with *seqrep*. If *seqrep* is relatively large and still a pattern is found with events that have a big time difference, then this pattern should get attention.

The time of the error event is displayed with the results. Time and date for the pattern's first appearance is provided with the results. An overwhelming amount of information would be displayed if time information should be provided for all matched sequences for

[1]In GigaSunet there are normally about 30 log entries per minute.

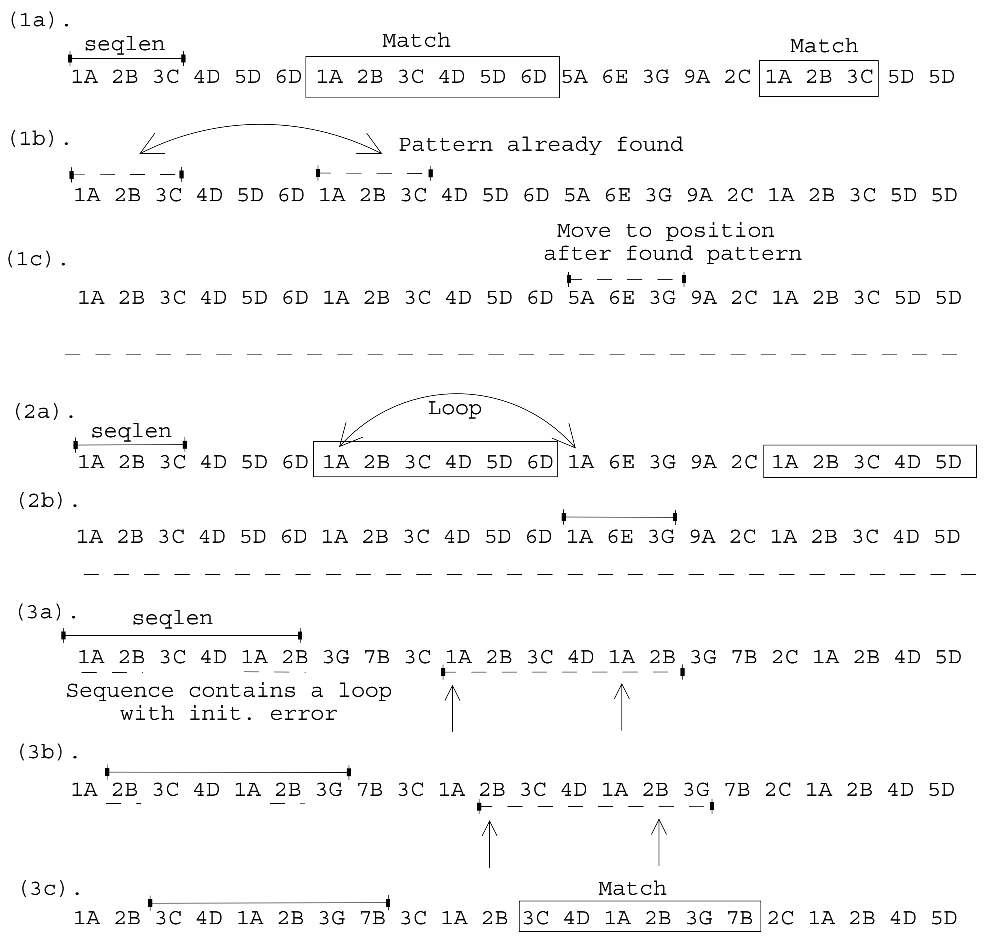

(1a).
seqlen
Match
Match
1A 2B 3C 4D 5D 6D 1A 2B 3C 4D 5D 6D 5A 6E 3G 9A 2C 1A 2B 3C 5D 5D
(1b).
Pattern already found
1A 2B 3C 4D 5D 6D 1A 2B 3C 4D 5D 6D 5A 6E 3G 9A 2C 1A 2B 3C 5D 5D
(1c).
Move to position after found pattern
1A 2B 3C 4D 5D 6D 1A 2B 3C 4D 5D 6D 5A 6E 3G 9A 2C 1A 2B 3C 5D 5D
(2a).
Loop
seqlen
1A 2B 3C 4D 5D 6D 1A 2B 3C 4D 5D 6D 1A 6E 3G 9A 2C 1A 2B 3C 4D 5D
(2b).
1A 2B 3C 4D 5D 6D 1A 2B 3C 4D 5D 6D 1A 6E 3G 9A 2C 1A 2B 3C 4D 5D
(3a).
seqlen
1A 2B 3C 4D 1A 2B 3G 7B 3C 1A 2B 3C 4D 1A 2B 3G 7B 2C 1A 2B 4D 5D
Sequence contains a loop with init. error
(3b).
1A 2B 3C 4D 1A 2B 3G 7B 3C 1A 2B 3C 4D 1A 2B 3G 7B 2C 1A 2B 4D 5D
(3c).
Match
1A 2B 3C 4D 1A 2B 3G 7B 3C 1A 2B 3C 4D 1A 2B 3G 7B 2C 1A 2B 4D 5D

Figure 3.15: Three scenarios when searching for patterns.

a pattern. However, the user can change this as program input and all sequences will be displayed with date and time.

Source code for the recurring sequence algorithm is provided in appendix B for the interested reader.

3.3.3 Low frequency errors

Finding low frequent network errors will help network operators to locate errors that could cause serious disturbance in the network traffic, despite from being rare. Experiments and implementation for a low frequent search algorithm for this purpose is directly connected to the theory presented in chapter 2.3.3.

Some low frequent errors occur regular in time, while others might report a few times in a short period of time, and never show again. A low frequent error that has a behavior of a regular recurring nature is considered to be more important to find and observe by KTHNOC. One reason is that a normal router reconfiguration might generate a special error message that is perfectly normal, but low frequent. And further, one isolated incident during some period is not considered by the algorithm as discussed in the low frequent theory.

Hence, adapting the algorithm to meet those requirements by providing adjustable constants and dynamical adaption to the environment was therefore essential for the given task. As will be shown, the low frequent theory provides possibilities to meet those demands with different solutions in practice.

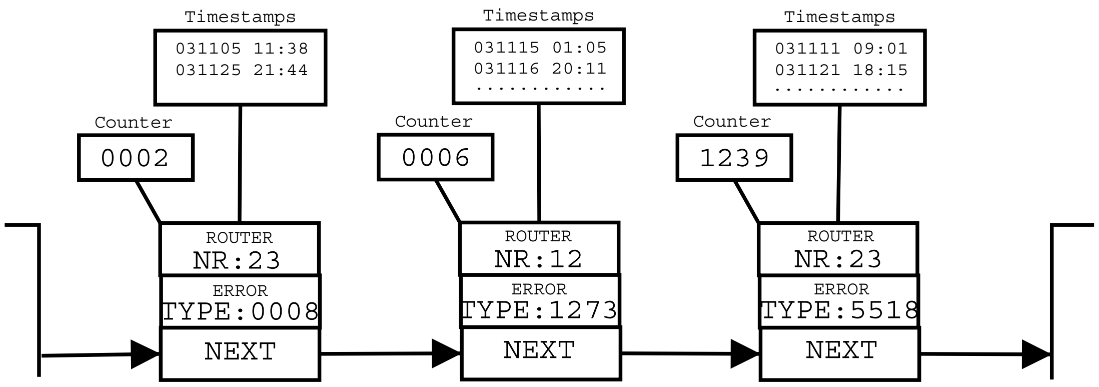


Figure 3.16: Cut from low frequent algorithm data storage structure.
Each router and error combined, here identified with numbers, will together represent a unique fault that is represented by a entry in the list.

Algorithm description

The implemented low frequent search algorithm begins with reading all available data from the database for the operator given time period. The entries collected from the database are inserted into a list were all errors are kept unique. That is, every entry in the list contains one and only one unique error and router combination.

Obtaining this structure is the first major task of the algorithm. This work requires considerable, but important work, since it provides for a suitable data structure for the forthcoming low frequent analysis. A schematic view of the data structure is shown in figure 3.16. The structure is implemented as a linked list and the code can be studied in appendix C for specific details.

As more elements of the same combination are found, a entry occurrence counter is incremented and a list of timestamps[2] is updated that belongs to the specific error combination. If the combination was not already present, a new entry will be created and

[2]Consequently the number of timestamps always match the value of the entry occurrence counter.

inserted into the list. As a result of this implementation, the processing run time will increase linearly[3] with the combinations in list.

The algorithm keeps track of the total number of rows captured from the database in the query set. From a calculation with this number, a limit is defined for a *low frequent definition*. This is done from computation with the build in constant LRFQFACT[4] as a factor multiplied with the number of rows present in the set. Hence, the error limit adjusts dynamically to the number of members in the set obtained by the query. One drawback with this solution is that experimenting with this constant requires a recompile of the program. However, tests so far have not indicated that fine tuning here is essential. The occurrence of an error is often very high, or very low. It is just a matter of finding a constant that separates the two extremes. Figure 3.17 shows how error entries from the list normally divide into different areas.

Only applying the above computation could be enough to reveal interesting low frequent errors in certain environments. But since specific conditions were presumed to be of special interest, this alone was not satisfactory. All entries in the algorithm list have to be examined for the low frequent definitions according to the theory described in chapter 2.3.3. All error combinations are evaluated with a boolean function that returns whether it is considered to be a valid error or not. This specific function, called *isValidPost*, processes the theory in practice. That means that all entries in the list shown in figure 3.16 pass through this function.

[3]There are other possible ways to solve this task. For example, a router by router query could decrease algorithm memory consumption, but will cause more SQL-queries on the database. The low frequent theory does not bind algorithm construction to one possible solution for obtaining and evaluating the data.

[4]Appendix C shows the details involved i C code. Built in constants are defined in the low frequent header file.

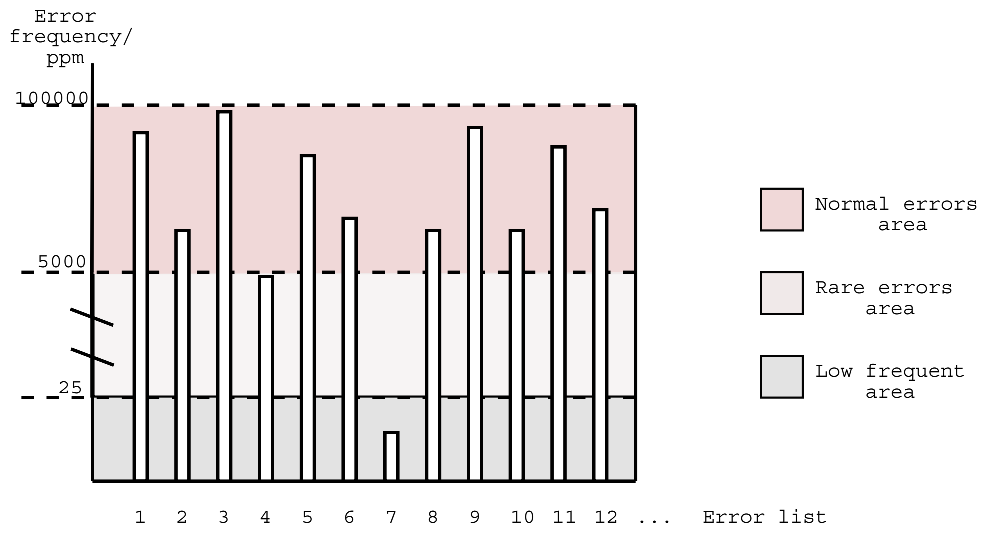


Figure 3.17: Error frequency areas.
Error and router combinations divide into different frequency areas from computing the error counter divided by the total number of error combinations in the set. The low frequent area is defined from the constant LRFQFACT. In this example set to 0.000025.

Algorithm details

As shown previously in figure 2.7 (page 30), all low frequent errors have some characteristic appearance in time. The algorithm needs to test the error regularity and the over all presence in time.

Two more built in constants are involved during this test. These are named PRESFACT and IREGFACT, for time period representation and regularity test respectively. Details referring constants and algorithm input variables will be discussed next.

The algorithm has both built in constants and input variables for algorithm computations and filtering. These are:

- Built in constants

  - Low frequent factor ($0 <$ LFRQFACT < 1): Is used by the algorithm to set an upper limit level to define what is to be considered as low frequent. This constant can be overrun from an algorithm input variable.

  - Time presence factor ($0 <$ PRESFACT < 1): Involves calculation to determine if the error is representing the chosen time period to a certain extent.
  - Error regularly factor ($0 <$ IREGFACT < 1): Involves calculation if the error behavior is of regular nature.

- Input variables
  - Time period: The algorithm search period start- and stop date.
  - Maximum level of errors: The algorithm will define a maximum level of errors found with the built in constant LFRQFACT. If this input variable is set to a higher level, the number of errors accepted as low frequent are increased. Hence, the constant LFRQFACT is overrun.
  - Error severity: Is defining the maximum error severity to report.

All built in constants are fixed at compile time. Experimenting will therefore require a recompile between each test run. Tuning constants for best performance is always a compromise between reporting irrelevant errors or excluding something important. Only from running real tests and analyzing the result those values could be set. Implications on this will be discussed further in chapter 4.3.3 and 5.2.

If the chosen time period exceeds one month, the algorithm must process two or more queries on the database since each month is represented with one table[5]. To make the upcoming computations more effective and to decrease memory consumption, a rough filtering[6] is processed for each table involved. All errors that pass this first test will be

[5]See earlier work[1] for details.

[6]This is done for the upper limit for the low frequent definition. That is, the number of rows for each table is computed with a constant(QURYFACT) setting temporary bounds for low frequent. This number

added to the data structure shown in figure 3.16. The algorithm is now ready to test all members in the list one by one according to the low frequent theory.

Function isValidPost computation

The theory presented in chapter 2.3.3 has been implemented almost straight forward into the algorithm in the boolean function called *isValidPost*. Each error combination from the list shown in figure 3.16 is evaluated in four steps by this function. These are as follows:

1. **Error severity test**.
   - Test the error severity grade compared to the value provided from the operator.
   - Involved parameter: Algorithm input argument severity grade.
2. **Low frequent definition test**.
   - Counts the number of error combinations and computes if the entry belongs in the defined low frequent area.
   - Involved parameter: Built in constant LFRQFACT.
3. **Interval presence test**.
   - Computes the representation in time from the operator given overall algorithm search period. Calculates the difference in time from the first and last appearance of the errors, and compares this value with the entire period.
   - Involved parameter: Built in constant PRESFACT.
4. **Irregularity test**.
   - Calculate if the error behaves in a regular recurring pattern to some extent.

is always set to a higher level than the overall level defined by LFRQFACT. This is needed because the number of rows differ in tables and the total number of errors is still not known at the time to set the final limit for low frequent.

- Involved parameter: Built in constant IREGFACT.

As soon as one of the above tests evaluates to *false*, the processing of the current entry is interrupted and the function will return *false*. Only if the entry passes all tests the function will return *true* and the entry will subsequently be reported.

The irregularity test has one special case. If the minimum of allowed errors occurs, two repetitions only, the evaluation is always true since two errors always represent a perfect regularity. Hence, the two repetitions entry will always pass the test. In practice, this will not cause any unwanted reports since such a rare combination that passes the previous tests, is always considered interesting.

Constants default values

All constant default values have been tuned with tests on real data. However, it is not possible to determine any perfect values. Networks are considered dynamic environments and repeated tests for constant tuning might be necessary over time as the network topology changes.

The following values have been determined as defaults for built in constants:

- LFRQFACT = 0.000025
- QURYFACT = 0.0003
- PRESFACT = 0.15
- IREGFACT = 0.67

Experimenting in this area will be presented in chapter 4.3.3 where the influence on error reports is shown.

3.4 Summary

In this chapter we have described the algorithms that were outlined in the background chapter, *Diffusion error detection*, *Recurring sequences* and *Low frequency errors*. All algorithms are based on analyzing captured network error data retrieved from the base system, described in 2.2.1. The *Diffusion* algorithm locates and reports errors that spread rapidly in networks. The *recurring sequence* algorithm finds patterns in logged router error reports. The *low frequency error algorithm* finds errors that are constantly recurring as an even stream over a long period of time.

Chapter 4

Results and Evaluation

4.1 Algorithm test outline

In this chapter results of the analysis of log data according to the algorithms outlined in the previous chapter will be presented and evaluated. Different time periods containing different amount of data will be used to measure the time for processing the data. These processing times are however only an indication, since they are only based on one run of a program. Several different values for the parameters in an algorithm will be used in presenting the results. As the research is divided into subareas result presentation and evaluation will be presented on each separate area.

For the diffusion algorithm, the time-window behaviour is investigated when searching for error diffusions. In the recurring sequence, patterns found will be matched against data in the database to verify its existence. For the low frequent algorithm, parameter adjustment takes place to produce less irrelevant data. Several tests are made on each algorithm and the results are presented.

This section summarises the results for the three implemented algorithms, *Diffusion error detection*, *Recurring sequences* and *Low frequent errors*.

4.2 Test environment

The programs were run on the same machine as the MySQL database. The time values for each area should only be a hint, none of the algorithms are optimized. Time values are based on one single run made on a Pentium III 600 MHz and 128 Mb ram.

4.3 Algorithm result and performance

All tests where performed on real network log data from the GigaSunet network. Detailed results from the implemented algorithms are presented separately for each algorithm tables, figures and diagrams. The main objective is to verify theory in practice but also to evaluate the overall algorithm performance and correctness at operation.

4.3.1 Diffusion error detection

The diffusion test runs were done month wise on three months, May, June and July. Tests on the first two months were done to see how much time difference there were on different amount of data. The result for May will be illustrated to see how the errors are found in a *time-window*. Table 4.1 gives a view of number of diffusions found for each selected month with used parameters.

For July different values for *time-window* and *report-level* were tested. At most 50 number of routers within 10 seconds were found twice. To verify this event a SQL query was made on the time-window for the last of the two known diffusions. This query revealed a total of 142 log entries in the time-window and 50 of them were %BGP-5-ADJCHANGE errors on 50 unique routers.

In the cases where the time-window is small the number of diffusions increases. At first glance this may seem like an error in the diffusion program. If a time-window of size 2 is chosen the window will move in steps of 1 second. If the window is bigger, say

Month	**Errors in month**	**time-window**	**report-level**	**Diffusions**	**Time**
May -03	354861	10	10	17	38 sec
June -03	404169	10	10	95	43 sec
July -03	479025	10	10	975	52 sec
July -03	479025	10	20	633	52 sec
July -03	479025	10	40	146	47 sec
July -03	479025	10	50	2	47 sec
July -03	479025	10	51	n/a	49 sec
July -03	479025	8	10	1080	52 sec
July -03	479025	6	10	1209	1 min
July -03	479025	4	10	1375	1 min 12 sec
July -03	479025	2	10	1493	1 min 48 sec
July -03	479025	2	20	475	1 min 46 sec
July -03	479025	2	30	116	1 min 44 sec
July -03	479025	2	40	44	1 min 43 sec
July -03	479025	2	45	14	1 min 46 sec
July -03	479025	2	46	6	1 min 44 sec
July -03	479025	2	47	1	1 min 45 sec
July -03	479025	2	48	1	1 min 44 sec
July -03	479025	2	49*	n/a	1 min 46 sec

Table 4.1: Test results for May, June and July.
n/a : Result not available. ***** : No result expected

10 seconds these errors would be caught the first time spotted in the window and then the time-window would move forward past several errors and hence the number of found diffusions would decrease. Scenarios like this require many errors in a small period of time. This was discussed in section 3.3.1. As seen in table 4.1 there was one diffusion with 48 routers all within 2 seconds. The reason why the last result outcome was known, marked with *, is that in the result the number of routers involved in a diffusion is displayed. This is shown in figure 4.1, *Number of routers in list.*

As pointed out earlier in chapter 3.3.1 this implementation has a drawback of possible missed[1] diffusions. In this implementation the *time-window* moves in discrete steps. The risk of missing diffusions are still considered small. Most diffusion cases studied have intense

[1]This was shown earlier in figure 3.9 (page 53)

bursts concentrated in a small time gap. Setting a to high algorithm *report-level* will be more significant in this matter. However, setting the *time-window* size to 2 seconds, the minimum value, will eliminate this phenomenon. This is due to the fact that logs also are recorded in discrete steps. Running the algorithm with minimum *time-window* size makes it slower and have the effects described above, but makes it failsafe for missed diffusions.

```
 ...
+------------------------------------------------------+
|                     Diffusion                        |
|------------------------------------------------------|
|      Error: %BGP-5-ADJCHANGE                         |
|      Diffusion error first at 05:28:34 03-05-13      |
|      Number of routers in list 13                    |
|------------------------------------------------------|
+------------------------------------------------------+
|                     Diffusion                        |
|------------------------------------------------------|
|      Error: %BGP-3-NOTIFICATION                      |
|      Diffusion error first at 05:28:35 03-05-13      |
|      Number of routers in list 12                    |
|------------------------------------------------------|
+------------------------------------------------------+
|                     Diffusion                        |
|------------------------------------------------------|
|      Error: %BGP-5-ADJCHANGE                         |
|      Diffusion error first at 05:28:36 03-05-13      |
|      Number of routers in list 22                    |
|------------------------------------------------------|
 ...
```

Figure 4.1: Output from the diffusion program.

Command: ./diffusion 2003-05-01 2003-05-20 10 10 cisco

One diffusion from May will be selected to verify that errors are found correctly in the time-windows and routers within the time-window are uniquely counted. Figure 4.1 shows a cut of 3 diffusions of a total of 17, from the diffusion program in the interval 2003-05-01 to 2003-05-31. Window size used was 10 seconds and the minimal number of routers 10.

The information in the program output will tell the error type, time and how many

```
mysql> select rindex,ltime,localdate,errorcode from ‘cisco-0305‘
> where ltime <= "05:28:35" and ltime >= "05:28:25"
> and errorcode = "%BGP-5-ADJCHANGE" and localdate = "2003-05-13";
     +--------+----------+------------+------------------+
     | rindex | ltime    | localdate  | errorcode        |
     +--------+----------+------------+------------------+
     |      9 | 05:28:34 | 2003-05-13 | %BGP-5-ADJCHANGE |
     +--------+----------+------------+------------------+
```

Figure 4.2: SQL query on the interval 05:28:25 - 05:28:35.

rindex : Router index, unique identifier.
ltime : Local time, time when the log row was archived.
localdate: Local date, date when the log row was archived.

routers involved in the diffusion. *Number of routers in list* provides the number of routers that had a certain error. This number is compared with *report-level* and if equal or more it is matched as a diffusion. Start looking at the first diffusion in the figure, *Error: %BGP-5-ADJCHANGE*. The diffusion starts at 05:28:34 2003-05-13 and has a total number of 13 routers in the current time-window. The window size set to 10 seconds gives the intervals for the time-window: 0-10, 5-15, 10-20 and so on. If the diffusion started at 05:28:34 the window must be either in the 05:28:25 - 05:28:35 interval or in 05:28:30 - 05:28:40.

To verify in which time-window the errors were found two SQL queries are run on the database. The first query on the 05:28:25 - 05:28:35 interval and the other 05:28:30 - 05:28:40. In figure 4.2 the SQL query shows that the 05:28:25 - 05:28:35 interval contains one %BGP-5-ADJCHANGE error. Errors must now belong to the latter time-window, 05:28:30 - 05:28:40. The SQL query and its result is shown in figure 4.3. To match the diffusion ten routers have to have the same error within the interval. There are thirteen unique rows in the figure and this verifies that errors were found in the 05:28:30 - 05:28:40 interval.

When the diffusion program has completed the search for diffusions, it displays its given parameters and summarizes the result, as displayed in figure 4.4.

```
mysql> select rindex,ltime,localdate,errorcode from ‘cisco-0305‘
where ltime >= "05:28:30" and ltime <= "05:28:40" and
localdate = "2003-05-13" and errorcode = "%BGP-5-ADJCHANGE";
+--------+----------+------------+------------------+
| rindex | ltime    | localdate  | errorcode        |
+--------+----------+------------+------------------+
|      9 | 05:28:34 | 2003-05-13 | %BGP-5-ADJCHANGE |
|      4 | 05:28:36 | 2003-05-13 | %BGP-5-ADJCHANGE |
|     23 | 05:28:36 | 2003-05-13 | %BGP-5-ADJCHANGE |
|     41 | 05:28:36 | 2003-05-13 | %BGP-5-ADJCHANGE |
|     49 | 05:28:37 | 2003-05-13 | %BGP-5-ADJCHANGE |
|     45 | 05:28:37 | 2003-05-13 | %BGP-5-ADJCHANGE |
|     33 | 05:28:39 | 2003-05-13 | %BGP-5-ADJCHANGE |
|     10 | 05:28:39 | 2003-05-13 | %BGP-5-ADJCHANGE |
|     22 | 05:28:39 | 2003-05-13 | %BGP-5-ADJCHANGE |
|     51 | 05:28:39 | 2003-05-13 | %BGP-5-ADJCHANGE |
|     24 | 05:28:39 | 2003-05-13 | %BGP-5-ADJCHANGE |
|     28 | 05:28:40 | 2003-05-13 | %BGP-5-ADJCHANGE |
|     30 | 05:28:40 | 2003-05-13 | %BGP-5-ADJCHANGE |
+--------+----------+------------+------------------+
```

Figure 4.3: SQL output, time-window 05:28:30 - 05:28:40.

```
+-----------------------------------------------------+
| Values used in this session                         |
|-----------------------------------------------------|
| Time period 2003-05-01 - 2003-05-20                 |
| Window size in seconds 10                           |
| Min number of routers in spread 10                  |
|-----------------------------------------------------|
| Summary                                             |
|-----------------------------------------------------|
| Number of diffusions in session 9                   |
+-----------------------------------------------------+
```

Figure 4.4: Diffusion summary window.

4.3.2 Recurring sequences

On the basis of results from runs of the recurring sequence program a table has been put together. Table 4.2 shows the result from the tests.

Row	Time period	Errors	seqlen	seqrep	Found patterns	Max rep.	Time
1	Jan -03	209541	10	10	72	36	2h 35m 00s
2	Feb -03	164823	10	10	109	208	1h 25m 00s
3	14-20 Feb -03	48856	12	10	n/a	n/a	0h 12m 30s
4	14-20 Feb -03	48856	11	10	9	19	0h 11m 44s
5	14-20 Feb -03	48856	10	10	13	23	0h 07m 00s
6	14-20 Feb -03	48856	10	23	1	23	0h 10m 45s
7	14-20 Feb -03	48856	10	24*	n/a	n/a	0h 10m 45s
8	14-20 Feb -03	48856	8	23	31	115	0h 04m 15s
9	14-20 Feb -03	48856	6	115	11	202	0h 03m 20s
10	14-20 Feb -03	48856	4	202	66	1146	0h 01m 30s
11	14-20 Feb -03	48856	100*	10	n/a	n/a	0h 10m 00s
12	14-20 Feb -03	48856	10	100*	n/a	n/a	0h 07m 00s

Table 4.2: Recurring sequence test runs results
Max rep : The maximal number of repetitions for a sequence. **seqlen** : The length of a sequence. **seqrep** : The number of repetitions for a sequence. **n/a** : Result not available. ***** : No result expected with used value.

As the first two rows show, it takes quite some time when the program is run over a hole month, depending of the amount of data stored for the time period. Runs over a longer time period could be done as a scheduled job once a month on an appropriate server.
By comparing the first two rows with others one can see that a month takes far more time than a week. Running the program over months however could reveal more patterns than week wise, since a sequence can start in an earlier week and continue into the next and so forth. Monthly runs can reveal less frequent patterns and shorter time periods only high frequent if the same *seqrep* parameter is used for the different time periods. Choosing a smaller *seqrep* for the shorter time period could produce similar result as a larger *seqrep* would on a longer period of time. This however requires that the pattern is recurring and found in the smaller time period.

To verify that the algorithm works correctly tests were done with values that should not produce any result. These are marked with a *. On the seventh row there is a marker at 24. In the previous runs the maximal number of repetitions for a pattern is 23. To verify this, test values shown at row five and six were used. As expected no result was produced for *seqrep* $= 24$.

For the week 14-20 in February a pattern with length 10 is found 31 times at the most in the time interval. Selecting a shorter sequence length for this week would most likely produce a result with a greater number of repetition. This is confirmed in the table as *seqlen* 8, 6 and 4 are chosen. With *seqlen* 8, 6 and 4 the previous run's maximal repetition value is used, it is known that there is at least one sequence that has repeated that many times. So choosing a shorter seqlen and previous maximal repetitions as seqrep should produce more or equal number of found patterns as the previous run. To verify that more patterns should be found, the maximal number of repetitions from the latter test is chosen. The number of repetitions increase when choosing a smaller sequence length. As seen there are many found patterns when choosing a small *seqlen*, sequence length. This scenario is somewhat expected as most of the patterns are random sequences created by mere accident. Depending on the amount of traffic and the size of the network a balance must be found between *seqlen* and *seqrep* to get reasonable patterns.

At row 5, the time for the week with *seqlen* $= 10$ and *seqrep* $= 10$ is less than the week with *seqlen* $= 100$ and seqrep $=$ *10*, row 11. This depends on the number of movements for the search sequence. As chapter 3.3.2 describes: if a sequence is found for a pattern the search sequence is moved forward *maxlen* steps. When a larger *seqlen* is chosen less patterns will be found and the search sequence will only move forward one step when no sequence can be matched as a pattern. With extreme large *seqlen* the time should decrease, but this would be unrealistic. The week with *seqrep* $= 100$ takes just as long as the one with *seqrep* $= 10$. After the algorithm has found patterns these must be evaluated if they

```
 Pattern [1] beginning with this sequence was repeated 10 times
Router:(7) kthnoc-2.sunet.se     %PIM-6-INVALID_RP_JOIN        [21:13:00 | 2003-02-19]
Router:(1) lulea1.sunet.se       %PIM-6-INVALID_RP_JOIN        [21:13:16 | 2003-02-19]
Router:(2) lulea2.sunet.se       %PIM-6-INVALID_RP_JOIN        [21:13:16 | 2003-02-19]
Router:(4) kthnoc-1.sunet.se     %PIM-4-DEPRECATED_HELLO_TLV   [21:13:24 | 2003-02-19]
Router:(3) stockholm2.sunet.se   %TCP-6-BADAUTH                [21:13:43 | 2003-02-19]
Router:(3) stockholm2.sunet.se   %TCP-6-BADAUTH                [21:13:43 | 2003-02-19]
Router:(1) lulea1.sunet.se       %PIM-6-INVALID_RP_JOIN        [21:14:19 | 2003-02-19]
Router:(2) lulea2.sunet.se       %PIM-6-INVALID_RP_JOIN        [21:14:19 | 2003-02-19]
Router:(4) kthnoc-1.sunet.se     %PIM-4-DEPRECATED_HELLO_TLV   [21:14:24 | 2003-02-19]
Router:(3) stockholm2.sunet.se   %TCP-6-BADAUTH                [21:14:47 | 2003-02-19]

 Pattern [2] beginning with this sequence was repeated 15 times
Router:(7) kthnoc-2.sunet.se     %PIM-6-INVALID_RP_JOIN        [18:13:25 | 2003-02-19]
Router:(3) stockholm2.sunet.se   %TCP-6-BADAUTH                [18:13:26 | 2003-02-19]
Router:(3) stockholm2.sunet.se   %TCP-6-BADAUTH                [18:13:26 | 2003-02-19]
...
```

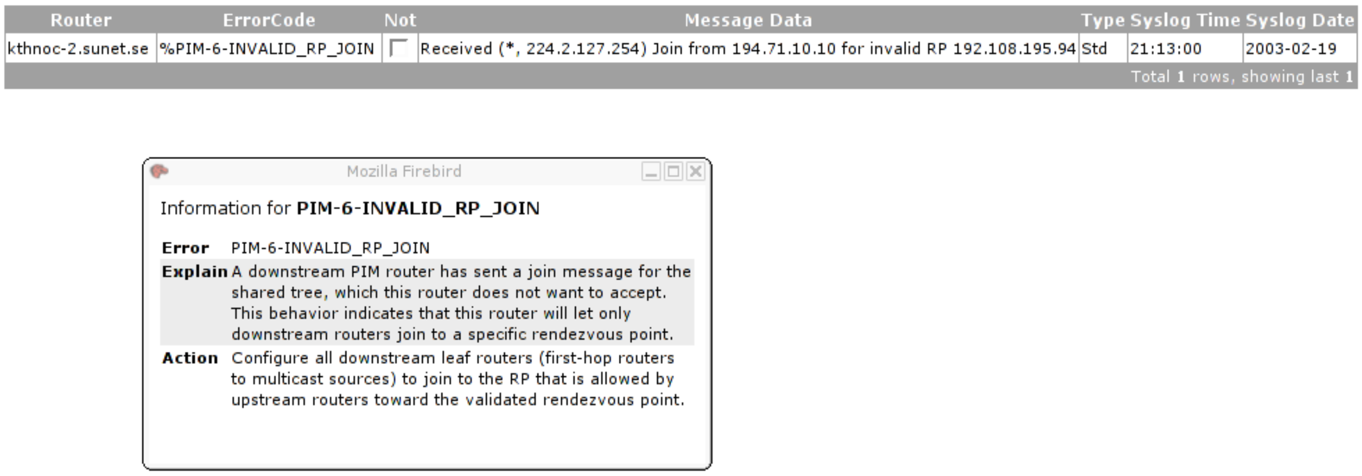
Figure 4.5: Program output run from a console

match the *seqrep* criteria. The amount of time in finding the patterns is equal but since in this case no result is to be displayed it takes less time than if result was to be displayed.

By default the program will only display the first occurrence of a pattern. Since hundreds of pattern can exist the result can be overwhelming. This can however be overridden by a program parameter and all of the found patterns will be displayed. A cut of a run with the default mode is shown in figure 4.5.

The first pattern found, **Pattern[1]**, in figure 4.5 begins with the shown sequence. All ten sequences for pattern[1] have matched to this sequence of length *seqlen*. Why it says: "begins with" is because as described in chapter 3.3.2 the algorithm will try to match more than *seqlen* errors. This means that the sequences not shown can be longer or equal to the shown pattern. An example of this is shown in figure 4.7.

Given the starting time for the pattern one can look up the sequence by either using the GUI provided by the base system[1] or using a database client software to get more information. In figure 4.6 the GUI has been used to look up the sequence to find out more about the starting router in the sequence. This is just a demonstration, and details about the actual information are not discussed.



Router	ErrorCode	Not	Message Data	Type	Syslog Time	Syslog Date
kthnoc-2.sunet.se	%PIM-6-INVALID_RP_JOIN	☐	Received (*, 224.2.127.254) Join from 194.71.10.10 for invalid RP 192.108.195.94	Std	21:13:00	2003-02-19

Total **1** rows, showing last **1**

Mozilla Firebird

Information for **PIM-6-INVALID_RP_JOIN**

Error	PIM-6-INVALID_RP_JOIN
Explain	A downstream PIM router has sent a join message for the shared tree, which this router does not want to accept. This behavior indicates that this router will let only downstream routers join to a specific rendezvous point.
Action	Configure all downstream leaf routers (first-hop routers to multicast sources) to join to the RP that is allowed by upstream routers toward the validated rendezvous point.

Figure 4.6: Using the GUI to retrieve more information about an error.

Running the algorithm over shorter time periods can result in missed sequences for a pattern. One day runs are not recommended as seen in figure 4.7 because earlier occurrences of a pattern can be missed. This pattern starts the 19:th of February and ends the 20:th, only the first three are shown in the figure. Running only on the 19:th will not produce any result.

Figure 4.7 shows three sequences for a pattern. The first sequence is the pattern which is searched for and the remaining are the sequences that matched to the first sequence. The output gives information about when the errors in the pattern were produced and from which routers. Source router and time for each sequence that matched the pattern gives enough information to look up the sequences in the database. With this information additional data can be extracted from the database.

Figures 4.8 and 4.9 are SQL queries on the database made on the first and last recurring

```
 Pattern [1] beginning with this sequence was repeated 10 times
 This is the first occurrence of this pattern.
 - The rest are listed in reversed order -
Router:(7) kthnoc-2.sunet.se     %PIM-6-INVALID_RP_JOIN        [21:13:00 | 2003-02-19]
Router:(1) lulea1.sunet.se       %PIM-6-INVALID_RP_JOIN        [21:13:16 | 2003-02-19]
Router:(2) lulea2.sunet.se       %PIM-6-INVALID_RP_JOIN        [21:13:16 | 2003-02-19]
Router:(4) kthnoc-1.sunet.se     %PIM-4-DEPRECATED_HELLO_TLV   [21:13:24 | 2003-02-19]
Router:(3) stockholm2.sunet.se   %TCP-6-BADAUTH                [21:13:43 | 2003-02-19]
Router:(3) stockholm2.sunet.se   %TCP-6-BADAUTH                [21:13:43 | 2003-02-19]
Router:(1) lulea1.sunet.se       %PIM-6-INVALID_RP_JOIN        [21:14:19 | 2003-02-19]
Router:(2) lulea2.sunet.se       %PIM-6-INVALID_RP_JOIN        [21:14:19 | 2003-02-19]
Router:(4) kthnoc-1.sunet.se     %PIM-4-DEPRECATED_HELLO_TLV   [21:14:24 | 2003-02-19]
Router:(3) stockholm2.sunet.se   %TCP-6-BADAUTH                [21:14:47 | 2003-02-19]
-----------------------------------------------------------
Router:(7) kthnoc-2.sunet.se     %PIM-6-INVALID_RP_JOIN        [20:08:57 | 2003-02-20]
Router:(1) lulea1.sunet.se       %PIM-6-INVALID_RP_JOIN        [20:09:05 | 2003-02-20]
Router:(2) lulea2.sunet.se       %PIM-6-INVALID_RP_JOIN        [20:09:05 | 2003-02-20]
Router:(4) kthnoc-1.sunet.se     %PIM-4-DEPRECATED_HELLO_TLV   [20:09:10 | 2003-02-20]
Router:(3) stockholm2.sunet.se   %TCP-6-BADAUTH                [20:09:46 | 2003-02-20]
Router:(3) stockholm2.sunet.se   %TCP-6-BADAUTH                [20:09:46 | 2003-02-20]
Router:(1) lulea1.sunet.se       %PIM-6-INVALID_RP_JOIN        [20:10:07 | 2003-02-20]
Router:(2) lulea2.sunet.se       %PIM-6-INVALID_RP_JOIN        [20:10:07 | 2003-02-20]
Router:(4) kthnoc-1.sunet.se     %PIM-4-DEPRECATED_HELLO_TLV   [20:10:10 | 2003-02-20]
Router:(3) stockholm2.sunet.se   %TCP-6-BADAUTH                [20:10:50 | 2003-02-20]
Router:(3) stockholm2.sunet.se   %TCP-6-BADAUTH                [20:10:50 | 2003-02-20]
-----------------------------------------------------------
Router:(7) kthnoc-2.sunet.se     %PIM-6-INVALID_RP_JOIN        [20:06:58 | 2003-02-20]
Router:(1) lulea1.sunet.se       %PIM-6-INVALID_RP_JOIN        [20:07:00 | 2003-02-20]
Router:(2) lulea2.sunet.se       %PIM-6-INVALID_RP_JOIN        [20:07:00 | 2003-02-20]
Router:(4) kthnoc-1.sunet.se     %PIM-4-DEPRECATED_HELLO_TLV   [20:07:10 | 2003-02-20]
Router:(3) stockholm2.sunet.se   %TCP-6-BADAUTH                [20:07:38 | 2003-02-20]
Router:(3) stockholm2.sunet.se   %TCP-6-BADAUTH                [20:07:38 | 2003-02-20]
Router:(1) lulea1.sunet.se       %PIM-6-INVALID_RP_JOIN        [20:08:02 | 2003-02-20]
Router:(2) lulea2.sunet.se       %PIM-6-INVALID_RP_JOIN        [20:08:02 | 2003-02-20]
Router:(4) kthnoc-1.sunet.se     %PIM-4-DEPRECATED_HELLO_TLV   [20:08:10 | 2003-02-20]
Router:(3) stockholm2.sunet.se   %TCP-6-BADAUTH                [20:08:42 | 2003-02-20]
Router:(3) stockholm2.sunet.se   %TCP-6-BADAUTH                [20:08:42 | 2003-02-20]
-----------------------------------------------------------
```

Figure 4.7: Cut of program output with *show all* option

[Only three of ten sequences are shown in the figure.]

```
mysql> select rindex,errorcode,ltime from ‘cisco-0302‘
where localdate = "2003-02-19" and ltime <= "21:14:47"
and ltime >= "21:13:00";
+--------+-----------------------------+----------+
| rindex | errorcode                   | ltime    |
+--------+-----------------------------+----------+
|      7 | %PIM-6-INVALID_RP_JOIN      | 21:13:00 |
|      1 | %PIM-6-INVALID_RP_JOIN      | 21:13:16 |
|      2 | %PIM-6-INVALID_RP_JOIN      | 21:13:16 |
|      4 | %PIM-4-DEPRECATED_HELLO_TLV | 21:13:24 |
|      3 | %TCP-6-BADAUTH              | 21:13:43 |
|      3 | %TCP-6-BADAUTH              | 21:13:43 |
|      1 | %PIM-6-INVALID_RP_JOIN      | 21:14:19 |
|      2 | %PIM-6-INVALID_RP_JOIN      | 21:14:19 |
|      4 | %PIM-4-DEPRECATED_HELLO_TLV | 21:14:24 |
|      3 | %TCP-6-BADAUTH              | 21:14:47 |
|      3 | %TCP-6-BADAUTH              | 21:14:47 |
+--------+-----------------------------+----------+
```

Figure 4.8: First occurrence for pattern [1].

sequences. Figure 4.8 is *pattern[1]*, first result from the recurring sequence program in figure 4.7. Figure 4.9 is the last sequence found for *pattern[1]*.

When the program has completed its processing and the result has been displayed a short summary is displayed. In the summary the length of the longest pattern is displayed. With this information it is not required to increase *seqlen* in a testing manner to get the longest pattern, assuming that the same *seqrep* is used.

4.3.3 Low frequent errors

Running the low frequent algorithm gives room for a lot of experimenting with input parameters and the built in constants. There will always be a compromise between producing too much and irrelevant data or to exclude relevant data. While testing has focused on the algorithm correctness, evaluation has also been carried out in the process.

When studying results from different test runs, it is clear that low frequent errors do

```
mysql> select rindex,errorcode,ltime from ‘cisco-0302‘
where localdate = "2003-02-20" and ltime >= "20:08:57"
and ltime <= "20:10:50";
+--------+-----------------------------+----------+
| rindex | errorcode                   | ltime    |
+--------+-----------------------------+----------+
|      7 | %PIM-6-INVALID_RP_JOIN      | 20:08:57 |
|      1 | %PIM-6-INVALID_RP_JOIN      | 20:09:05 |
|      2 | %PIM-6-INVALID_RP_JOIN      | 20:09:05 |
|      4 | %PIM-4-DEPRECATED_HELLO_TLV | 20:09:10 |
|      3 | %TCP-6-BADAUTH              | 20:09:46 |
|      3 | %TCP-6-BADAUTH              | 20:09:46 |
|      1 | %PIM-6-INVALID_RP_JOIN      | 20:10:07 |
|      2 | %PIM-6-INVALID_RP_JOIN      | 20:10:07 |
|      4 | %PIM-4-DEPRECATED_HELLO_TLV | 20:10:10 |
|      3 | %TCP-6-BADAUTH              | 20:10:50 |
|      3 | %TCP-6-BADAUTH              | 20:10:50 |
+--------+-----------------------------+----------+
```

Figure 4.9: Last occurrence for the first pattern.

exist and can be found. To tune the algorithm built in constants for best performance, tests were carried out for the period Jan 03 - Aug 03, monthly.

At present the algorithm produces all results on *stdout*. A typical report from one month produces about two to twenty entries. Figure 4.10 shows one of the found entries from the February run as an example of the program output. There were two occurrences of the error %GRP-3-COREDUMP[2] in router *vasteras2.sunet.se*. Along with the above information the program output[3] provides data for the error occasions.

One month of testing at the time seemed to be a reasonable assumption for a normal low frequent error search. However, running the algorithm over a longer time period does not necessary increase the number of reports, since the algorithm adapts dynamically. For

[2]Evaluation on the significance of the incident requires expertise and additional data.

[3]If the program output in the future is inserted into a database for trend analysis, comparing test runs might be performed better. At this stage the amount of data to analyze easily gets to large for human inspection. Further development in this area will be discussed in chapter 5.3, Future work.

```
+-------------------------------------------+
|             LOW frequent ERROR            |
+-------------------------------------------+
| Error    : %GRP-3-COREDUMP                |
| Router nr: 49                             |
| Router   : vasteras2.sunet.se             |
| Num hits : 2                              |
+-----------------+-------------------------+
|    Foundlist    |       Timestamps        |
+-----------------+-------------------------+
| Num 1           | Sat Feb 15 09:58:30 2003 |
| Num 2           | Sun Feb 23 01:14:38 2003 |
+-----------------+-------------------------+
```

Figure 4.10: Low frequent entry printout.

For all low frequent error a text window is displayed. The window contains the information error code, router and time. Foundlist contains a list of each occurrence mapped to the point in time when the error occurred. The displayed router is the component that reported the error.

example, if one low frequent candidate only showed within two weeks time and the chosen period is six months, it will not represent the total time period sufficient to pass the time period representation test. This must be tested and evaluated further in the future and the existence of the involved test must also be questioned from this reason.

A search extending up to six months or more is not considered to be unrealistic. But the process memory consumption might become a limiting factor. The memory usage is somewhat linear to the amount of errors investigated and one error consumes about 100bytes[4] of memory. E.g., five million errors will consume about 500MB memory and so forth.

Table 4.3 shows results from experimenting with the values of constants. Note how the time period representation factor[5] (PRESFACT) always has a big influence on the result.

[4]The memory consumption has been observed during test runs.
[5]See page 29, equation (2.1) for background theory.

Month	**Errors**	**Hits (PRESFACT)**				**IREGFACT**	**Dyn limit**	**Severity**	**Avg. time**
		0.05	**0.15**	**0.45**	**0.65**				
Jan -03	209541	24	16	6	3	0.67	5	6	0m25s
Jan -03	209541	25	16	6	3	0.75	5	6	0m24s
Feb -03	164823	36	35	26	1	0.67	4	6	0m16s
Feb -03	164823	64	63	26	1	0.75	4	6	0m19s
Feb -03	164823	22	21	12	0	0.67	2	6	015ms
Feb -03	164823	22	21	12	0	0.75	2	6	017ms
Mar -03	296690	4	3	2	0	0.67	7	6	0m30s
Mar -03	296690	4	3	2	0	0.75	7	6	0m31s
Apr -03	439575	39	32	25	9	0.67	10	6	0m57s
Apr -03	439575	45	38	29	10	0.75	10	6	1m
Apr -03	439575	37	30	23	7	0.67	5	6	0m48s
Apr -03	439575	43	36	27	8	0.75	5	6	0m51s
May -03	354861	18	15	5	3	0.67	8	6	0m33s
May -03	354861	19	15	5	3	0.75	8	6	0m34s
Jun -03	404169	15	12	8	5	0.67	10	6	0m42s
Jun -03	404169	21	18	14	7	0.75	10	6	0m40s
Jul -03	479025	10	10	2	0	0.67	12	6	0m47s
Jul -03	479025	13	13	5	2	0.75	12	6	0m46s
Aug -03	933059	20	19	10	0	0.67	25	6	2m
Aug -03	933059	22	20	11	0	0.75	25	6	2m

Table 4.3: Low frequent example results.

Severity: Highest grade of severity for an error. **Hits**: Found low frequent errors. The lower values are the PRESFACT constant, for time representation. **Dyn limit**: Dynamic limit, the upper limit for errors to define as low frequent. This is calculated from constant LFRQFACT. **IREGFACT**: Factor for maximum irregularity of errors. A lower value will give errors of higher regularity than a higher value, $0 <$ IREGFACT < 1.

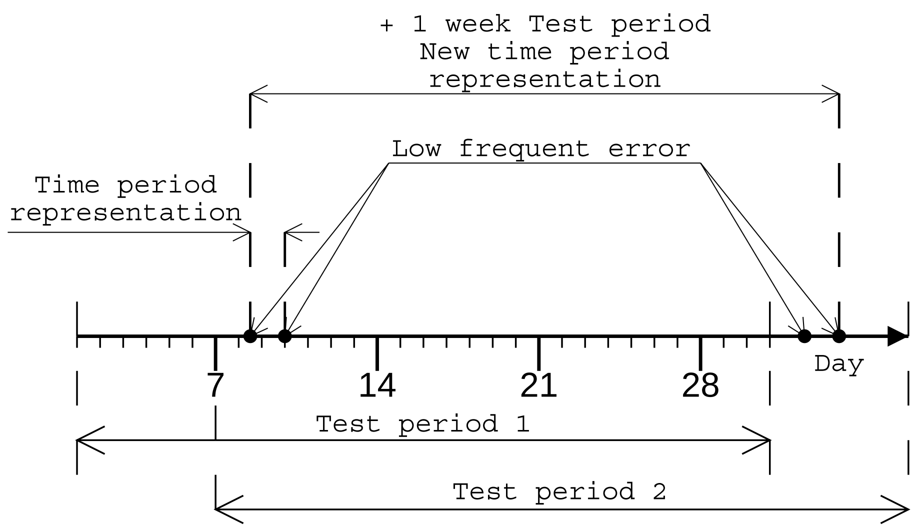


Figure 4.11: Low frequent time period representation.
Depending on the location of the event in time, the error time period representation can change drastically. In this example, moving the time window forward one week from period one to two will capture all events and therefore extend the time representation period to pass the test involving factor PRESFACT. If not, the error is disregarded as an isolated event and is therefore not reported.

Figure 4.11 better explains what happens and why the test itself should be questioned since it might remove interesting cases depending on the program usage. The figure shows how a single run could give too short time representation if the next event is outside the current test period. But changing the test period as a moving time window will report the error when the next event also gets inside. Hence, program usage is at present best suited for example a weekly job[6] set to investigate data from the previous thirty days.

The regularity test only has an influence on the result if some events were irregular to some extent. Due to this a constant change might not produce a different result. For example as shown in table 4.3 for the *Mar -03* test.

[6]A cron job is well suited for the task. A script example can be studied in Appendix E. It has a time period of one month and is scheduled to run once a week.

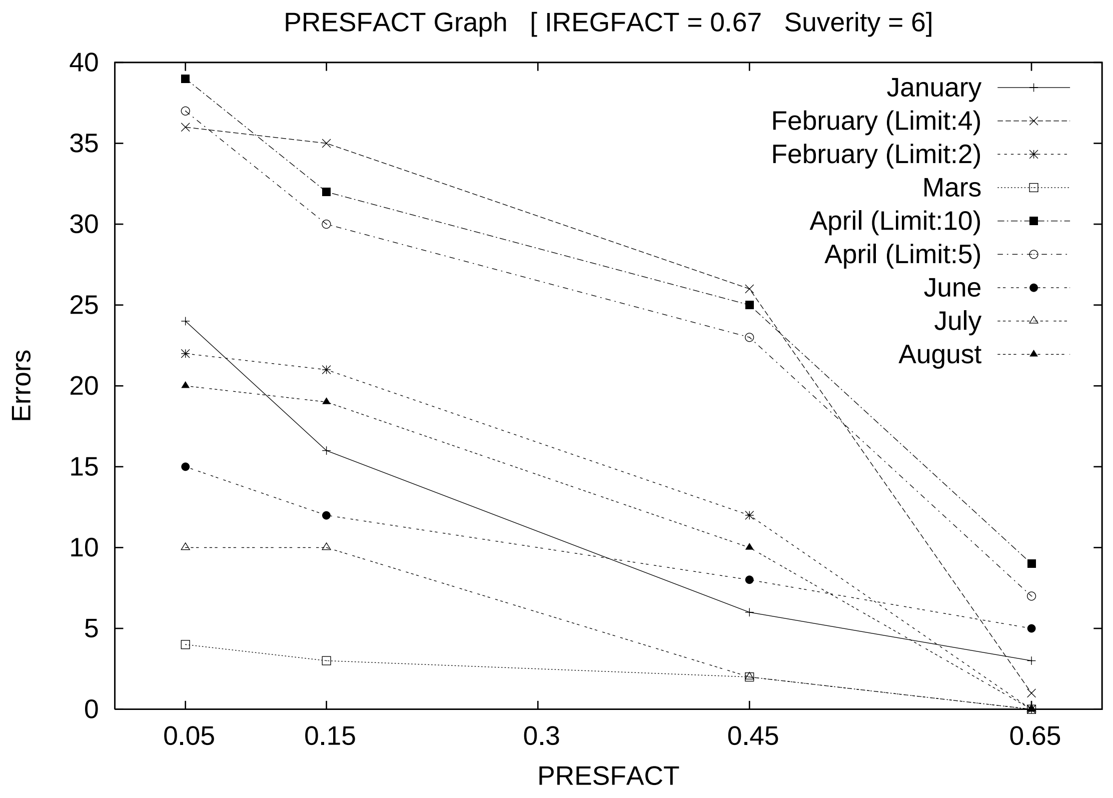


Figure 4.12: Low frequent error result graph.

The graph shows how the different values for the time period representation factor PRESFACT affects the number of reported low frequent errors.

Choosing a too low IREGFACT could result in not finding any events at all. Suitable values have proved to be about 0.6 - 0.7.

The value 0.67 was chosen after extended testing as it often proved to produce relevant data without dismissing any interesting cases found so far. After fixing this constant repeated tests were done for the time period representation. The graph 4.12 shows how different values for PRESFACT affect the outcome for low frequent errors found. Notice the sometimes accelerating effect from 0.45 to 0.65 which results in far less reported errors. E.g., observe how the *February (Limit 4)* run drops to one event only for this change of

```
Total number of error in set         = 164823
Number of low frequent errors found = 36
Upper limit for hits was set to      = 4
Max grade of severity was set to     = 6

Lowfreq LFRQFACT is set to           = 0.000025
Presens of total time PRESFACT       = 0.050000
Factor of max irregularity IREGFACT  = 0.670000
```

Figure 4.13: Low frequent program output summary.

Summary text shows the used values in the session. Upper cased words are built in constants. These constants are declared in lowfreq.h. Changes on these values require a recompile of the program. The grade of severity is operator given as a program input argument.

constant value. This means that almost all low frequent candidates occurred within two weeks time during this month.

The low frequent definition constant LRFQFACT did not require any extended analysis and experimenting at present. The reason is, as was discussed in theory, that the occurrence of an error event often is high. In that respect low frequent errors are extreme values. Setting this constant is therefore a task of approximating a level that is near to the extreme values. Testing indicated that 25 ppm was a well working limit. For example, in February -03 the limit will be $0.000025 * 164823 \approx 4$. Hence, only four repetitions are allowed to be considered as a low frequent error. Table 4.3 shows this value in column *Dyn limit* for the tested months.

The limit can be adjusted from a number provided in the program argument list. If this number is greater than the program calculated value it will count. It might not have any other practical use than to serve as a testing feature.

When the low frequent program completes it also displays a short summary of the values and results for each session. Figure 4.13 shows the program summary. Its main purpose is to serve as a basis for collecting statistical data for evaluation and tuning of constants.

4.4 Algorithm evaluation

Evaluation will be on how well the algorithm work against the issues laid out in chapter 2 and the result in the previous section. The impact of the findings given by the algorithms are not considered in detail in this master thesis and it is up to the responsible at the NOC to draw further conclusions.

The completion time for each of the program parts was something that never was discussed in the previous chapters. The recurring sequence algorithm takes much more time for its calculations compared to the two other programs as seen by test runs. This depends on its linear search method.

Parameters for each algorithm used in the previous chapter will need to be set more accurately by experienced network supervisors. The used values are mainly for testing purpose and should be considered as default values and specific for SUNET. Parameter values producing too much information as result are not wanted. The result should be a reasonable amount to handle and not overwhelming.

Diffusion error detection

Error diffusions in the network can be found with the diffusion algorithm as seen in the previous section. As shown in the previous section errors can spread to many routers within only a few seconds. In the shown tests, up to 48 different routers reported the same error within a two second time interval for July 2003. Detecting these events was the goal with this algorithm and it has proved to work satisfactory.

To get the most extreme cases of diffusion the time-window should be set for two seconds and the report level about 40-50 close and up to the maximal number of routers in the network. Having a time-window higher than two and a report level less than 40 will produce a result that has too many error diffusions to be manageable .

Recurring sequences

With the developed algorithm recurring sequences of error logs can be found, as shown in the previous section. The implemented algorithm finds patterns and information is provided for further investigation of the error data. Some of the found patterns are created by mere accident, but by setting the condition *secrep* patterns must be recurring and most or all of the haphazard patterns can be filtered out.

Having a sequence length of 10, which is about a fifth of all routers in SUNET, for a week is a reasonable sequence length. Too short sequence length will more likely produce haphazard patterns. The seqrep parameter, the number of repetitions, should be set around 10-15, not lower, to produce manageable results.

A drawback with the algorithm is the time for large time intervals as seen in table 4.2. This is not actually not really a drawback, since it is intended to run at a user selected time point, preferably after work hours.

Low frequency errors

The algorithm was successfully implemented from theory and is also working as expected. Its processing effectiveness is satisfactory but it has unexpected high memory consumption, something that becomes a limiting factor when running on long time periods.

Repeated tests have proved that low frequent errors commonly exist and can be found. The existence of low frequent errors were known by the NOC, but not to what extent. The errors have proved to be more common than expected and this often results in too many errors suitable for human inspection. The majority of the reported errors are also considered as ”not important”. The process of finding important errors has proved to be a harder task than first assumed, mostly due to the risk of excluding interesting cases if setting constants too restrictive.

For this case the input argument for severity proved to be useful. Most unwanted reported errors were of low[7] severity grade. This was partly expected but could not be taken for granted. As shown earlier in table 4.3, the used severity grade was six during all tests. Changing the minimum severity grade from six to three resulted in a significant improvement in reporting only relevant[8] data.

At this stage constants are tuned to not produce too much output since the network technicians need to read all data output manually. An exception was made for PRESFACT which needs more evaluation on real tests. If automated insertion into a database was to be implemented, some constants might prove to be unnecessary since they could be replaced with selective queries on the database. Extended possibilities as such will be discussed further in chapter 5.3.

4.5 Summary and information

In this chapter results for the implemented algorithms have been shown and evaluated. Different parameters for each algorithm have been discussed, and their impact on the result.

[7]A low grade of severity means a higher figure. The severity is defined from 0 - 6 by Cisco.

[8]This statement origins from experience at the NOC.

All algorithms operate according to the theory presented in chapter two and produced results that could be verified against data in the database. Further conclusions will be presented in the next chapter.

Chapter 5

Conclusion

5.1 Thesis conclusion and evaluation

In this master thesis we have focused on extended methods of finding troubleshooting information from the archived network log data. The idea was to search for special types of logs and log patterns and develop algorithms for locating these patterns. It has proved to be successful and the log data proved to conceal unknown error states. Uncovering the error states can be useful in maintaining the network and assists network technicians in fault detection.

Error diffusion was detected on several occasions. A diffusion is always considered as a serious fault which requires a network technician's attention. The diffusion algorithm finds and reports events recorded in the log data. The algorithm is best suited for daily or hourly runs since those events could require immediate attention. The algorithm is regarded as a successful tool for this purpose.

Recurring sequences were found to be related to the diffusion error theory since the error starting the sequence originates from one router and spreads to others. The algorithm implemented for this purpose has proved to work, but requires a great deal of CPU during operation. Pattern searches proved to require extensive work since a lot of comparisons must be made. Recurring sequences proved to exist in the log data, however their meaning and significance is still not known. The assumption was that, if the sequence originator could be located and fixed, the log data generation could be reduced as well as fixing a presumptive fault. This has not yet been proved.

The program finds patterns of various length but many should be considered as random sequences constructed by mere accident. To insure a higher level of correctness, i.e. that the found patterns actually are patterns created from a fault that spread the pattern must be repeated a reasonably number of times, within the search time period. As the length of the sequence also is significant, choosing a too short sequence length proved to give many unwanted false patterns.

Low frequent errors were located and proved to be more common than expected. This leads to problems in selecting the error types that KTHNOC considers relevant. Setting hard constrains in for example regular behavior always results in a risk of missing some interesting cases.

The low frequent program will need to be tuned for the environment it is intended to run in or else it can give misguiding result. If values for a large network are used in a smaller one it will result in no low frequent errors. Hence, various parameters must be set accordingly to the network in hand.

The link failure investigation was not implemented. The reasons were that such a proposed system would require supplementary data collected with SNMP and would also require a great deal of work. The area is of both technical and financial interest for WAN organisations since such a system can reduce their costs in hiring network links. The conclusion is that such a system might be possible to implement, but it is not trivial and without risk of unknown problem areas appearing in the process.

Solutions by using the error logs to build a state machine for SUNET in link failure detection have been investigated and discussed. By using state machines link failures could be more reliably detected. The investigation of state machines showed that the states can not be constructed and maintained only from using the error logs. SNMP is needed for constructing state machines for the network connectivity. By reading log data it can then be possible to make state transitions. As was stated in the link failure chapter using error logs is not entirely reliable as the link for the error message could be down. The current base system has no support for this kind of event.

Another investigation in this thesis is made on log deviation, detecting error log increase. It became clear that this can just as well be detected by using SNMP. It has also been stated by staff members at KTHNOC that large increase in the logging of errors is often generated by themselves upon network configuration.

We have proved that extended use of WAN log data is possible. By extensive use of this data troubleshooting assistance can be improved. We have successfully proved that diffusion, recurring sequences and low frequent errors exist in the log data. By using the implemented algorithms these errors can be found with a minimal personnel requirement by the NOC.

5.2 Problem areas

For all implemented algorithms the reports are presented on STDOUT. Running the algorithms on a regular basis will soon give a considerable amount of data to attend to. The conclusion is that a production system needs to handle the produced data automatically. E.g. automated insertion to a database would work well and open the possibility for achieving statistical reports etc.

Built in constants in algorithms were tuned to suit the specific environment in SUNET network. This is especially critical for the low frequent algorithm where three important constants are built in the algorithm source code. There are no further tests done on other environments to verify if constants are properly set or have to be returned for each network. Hence, there can be no conclusions at this point if this design is satisfactory when moving to other environments.

The main objective was not to evaluate the log data from algorithm reports in this thesis. But adjusting constants and improving algorithm functionality presented problems in the evaluation of relevant data from reports since this requires technical expertise at hand. Thus, without detailed knowledge of fault management in WANs, algorithm improvements sometimes get problematic. Better understanding of the algorithm output could have helped to move in the right direction and inspire new ideas.

5.3 Future work

Link failure detection by using error logs and SNMP was described in chapter 2.3.5. To limit the amount of work in this master thesis this feature has been disregarded. Providing a system for link failure detection is probably a thesis or bachelor project itself.

One area that has not been considered further in the thesis is trend analysis. Saving program output from the program parts either in a database or plain text files will give this possibility. Especially in the low frequent area. With this data it would be possible to follow the history of certain errors. Graphs for data presentation could be produced for error reports, as the example shown in figure 5.1.

This kind of graphs could be made by using the same database as the base system for

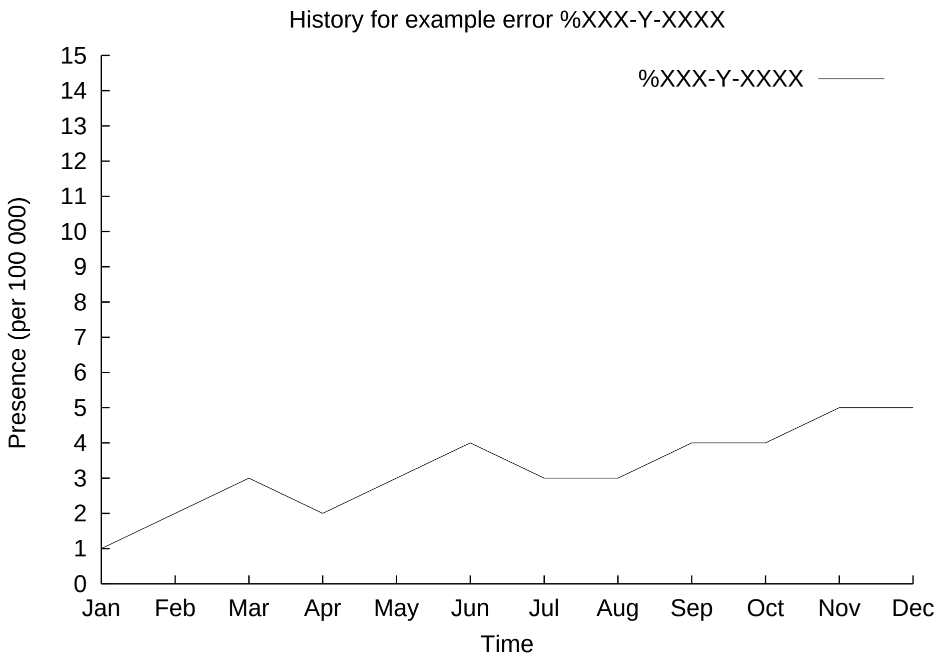


Figure 5.1: Example of a trend graph.

Fiction history of the error %XXX-Y-XXXX.

storing program output. PHP[1] could serve as a middle layer for retrieval and logic together with HTML for data presentation. Creating graphs is possible with PHP and there are free libraries for PHP when creating more advanced graphs[2].

5.4 Algorithm results from KTHNOC

In every day wide area network surveillance and troubleshooting by KTHNOC the low frequent program has proved to be helpful. KTHNOC has found a failure in one of the core routers in SUNET. The failure was found in one router and as the failure showed to be an operating system bug all of SUNET's core routers are affected as all run the same

[1]PHP: Hypertext Preprocessor. See http://www.php.net

[2]JpGraph is an OO class library for graphs using PHP. http://www.aditus.nu/jpgraph/index.php.

OS[3].

The failure was found by running the program to process data from the previous week. Further investigations led to the conclusion that there was a failure in the operation system[4]. This failure is now reported to Cisco and KTHNOC is waiting for further notice from Cisco. When or if a bug fix comes from Cisco all of SUNET's routers will need to be upgraded.

[3]OS version 12.0(23)s. The platform may differ but the OS is the same.
[4]Investigation and conclusion were made by KTHNOC

References

[1] *A log archiving system for Wide Area Networks* Böhm, Skantz KaU: 2003:01

[2] A. Ward , P. Glynn and K. Richardson, *Internet service performance failure detection*, Performance Evaluation Review, August 1998.

[3] C.Hood and C.Ji *Proactive network fault detection* in Proceedings of IEEE INFOCOM '97, Kobe Japan April 1997.

[4] *GPL license* http://www.gnu.org/licenses/gpl.html#SEC1

[5] I.Katzela and M.Schwartz, *Schemes for fault identification in communications networks*, IEEE/ACM Transactions on Networking, vol3(6), pp 753-764, December 1995

[6] *NTop* Free network surveillance tool. http://www.ntop.org/

[7] Paul Barford, Jeffery Kline, David Plonka and Amos Ron. A Signal Analysis of Network Traffic Anomalies. In *Proceedings of the second ACM SIGCOMM Workshop on Internet measurment workshop*, pages 71-82. ACM Press, 2002.

[8] Paul Barford and David Plonka. Characteristics of Network Traffic Flow Anomalies. In *Proceedings of the first ACM SIGCOMM Workshop on Internet measurment workshop*, pages 69-73. ACM Press, 2001.

[9] *PhpMyLog* GUI for Mylogd - a log capturing system.
http://iib.pilsnet.sunet.se/phpMyLog/

[10] R. Cáceres, *Measurements of wide area Internet traffic*,Tech.Rep. UCB/CSD 89/550, Computer Science Department, University of California, Berkeley, 1989.

[11] *UNIX Network Management* Steve Maxwell, McGraw-Hill, March 1999, ISBN 0079137822

[12] V. Paxson, *Measurements and Analysis of End-to-End Internet Dynamics*, Ph.D thesis, University of California, Berkeley, 1997.

[13] V. Paxson, *Fast, approximate synthesis of fractional gaussian noise for generating self-similar network traffic*, Computer Communications Review, vol. 27(5), pp 5-18, October 1997.

[14] F. Feather, D. Siewiorek and R. Maxion, *Fault detection in an ethernet network using anomaly signature matching* in Proceedings of ACM SIGCOMM '93, San Francisco, CA, September 2000.

Appendix A

Diffusion files

A.1 diffusion.c

```
/***************************************************************************
                          diffusion.c  -  description
        ----<<0>>--- * Detect error diffusion *  ---<<0>>---
                             -------------------
    begin                : Mon Feb 24 16:00:00 CET 2003
    copyright            : (C) 2003 by Mikael Bï£¡m & Henrik Skantz
    email                : micko@linuxmail.org, hesk@linux.se
***************************************************************************/
/***************************************************************************
 *   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 of the License, or     *
 *   (at your option) any later version.                                   *
 ***************************************************************************/
#include "header.h"

int main (
  int argc,
  char *argv[])
{

  if (argc != 6)
  {
    printf (USAGEMESS);
    exit (1);
  }

  /*
   * Start diffusion
   */
  diffusion (argv);
  return 0;
}

/*
 * Detects error diffusion
 */
void diffusion (
  char **in)
{
  char *sqldiffquery;
  char *dbtable;
  unsigned int listSize = 0;
  unsigned int winSize = 0;
  unsigned int minDiff = 0;
  unsigned short int nrDiffs = 0;
  MYSQL_POST *logList = NULL;

  Config *C1 = (Config *) malloc (sizeof (Config));
  winError *rp_cur = NULL;

  /*
   * Create the list containing posts inside sliding window
   */
  winEhead = mk_winError_post (NULL);
  rp_cur = winEhead;

  /*
   * Read the mylogd config file
   */
  readConf (CONFFILE, C1);      /* Read config to C1 */

  winSize = (unsigned) atoi (in[3]);
  minDiff = (unsigned) atoi (in[4]);

  /*
   * Connect to database
   */
  if (connectDB (C1) == NULL)
  {
    fprintf (stderr, "Connection to database failed!\n");
    terminator ();
  }
  /*
   * Set time from inargs
   */
  initTimeObjs (in[1], in[2]);

  /*
```

```
   * Set table from the last date from choosen period
   */
  if ((dbtable = setdbtable (in[2], in[5])) == NULL)
  {
    fprintf (stderr, "Could not set dbtable!\n");
    terminator ();
  }

  /*
   * Create an sqlquery to get data: (For one table only)
   */
  /*
   * [rindex, ltime, localtime, errorcode]
   */
  if ((sqldiffquery = diffSqlQuery (dbtable)) == NULL)
  {
    fprintf (stderr, "Could not create SQL-query!\n");
    terminator ();
  }
  /*
   * Mysql get complete interval start/end date
   */
  logList = getList (sqldiffquery);
  dp = initHackPointer ();      /*Pointer for comparing time-window entries */
  /*
   * Roll until all diffusions are found
   */
  while (difftime (mktime (fr_), mktime (to_)) != 0)
  {
    /*
     * Get post inside the timewindow
     */
    listSize = sumErrors (winSize, rp_cur);

    /*
     * Look for diffusion if current list has enough members
     */
    if (listSize >= minDiff)
    {
      nrDiffs += findDiff (winEhead, &minDiff);
    }
    /*
     * Move the window forward half its size
     */
    incr_sec (winSize / 2);
    /*
     * Prepare for next time-window
     */
    winEhead = removeList (winEhead);
    rp_cur = winEhead;
  }

  /*
   * Remove all entries in the global list of posts
   */
  remSQLtable ();
  free (dbtable);
  free (C1);
  free (sqldiffquery);

  display_summary (in, winSize, minDiff, nrDiffs);
}

/*
 * displays summary window.
 */
void display_summary (
  char **in,
  unsigned int winSize,
  unsigned int minDiff,
  unsigned short int nrDiffs)
{
  printf ("  _____________________________________________________\n");
  printf (" | Values used in this session                         |\n");
  printf (" +-----------------------------------------------------+\n");
  printf (" | Time period %s - %-27s|\n", in[1], in[2]);
  printf (" | Window size in seconds %-29d|\n", winSize);
  printf (" | Min number of routers in spread %-20d|\n", minDiff);
  printf (" +-----------------------------------------------------+\n");
  printf (" | Summary                                             |\n");
  printf (" +-----------------------------------------------------+\n");
  printf (" | Number of diffusions in session %-20d|\n", nrDiffs);
  printf (" +-----------------------------------------------------+\n");
}

/*
 * Put all errors  inside a time-window into the winError list.
 * toffset is a global reference to the list containing
 * retrieved data from the MySQL database.[listCommon.h]
 * Returns: the number of errors. Size of winError list.
 */
int sumErrors (
  int winSize,
  winError * rp_head)
{
  MYSQL_POST *cur;
  time_t postStart = 0;
  time_t winStart = 0;
```

```
int listSize = 0;
boolean hasOffset = false;
winError *rp_cur;
cur = toffset;                /* Get current offset in list */
winStart = mktime (fr_);      /* Get window starttime */
rp_cur = rp_head;

if (cur != NULL)
{
  postStart = getUTC (cur);   /* Get time of start-post */

  /*
   * Dont get any post that's before the time-window
   */
  if (postStart <= winStart)
    toffset = cur->next_post;
  /*
   * Roll until current post is outside the time-window
   */
  while (postStart <= (winStart + winSize))
  {
    rp_cur = mk_winError_post (rp_cur);       /* Add the post to end at list */
    rp_cur->mysqlPost = cur;
    listSize++;

    /*
     * Check if end of list, finished
     */
    if (cur->next_post == NULL)
      break;
    /*
     * Move on to next post in the MySQL data list
     */
    cur = cur->next_post;
    /*
     * Get the timestamp for next post
     */
    postStart = getUTC (cur);

    /*
     * Set post to start with next time
     */
    if ((postStart >= (winStart + (winSize / 2))) && (hasOffset == false))
    {
      toffset = cur;
      hasOffset = true;
    }
  }
}
```

```
  else
    listSize = 0;

  return listSize;
}

/*
 * Locate any possible diffusion inside time-window
 * If located, error and time is displayed
 *
 * Returns: nrDiffs the number of diffusions
 */
int findDiff (
  winError * head,
  unsigned int *minDiff)
{
  diffList *diffHead;
  diffList *curDiff;
  winError *rp_cur;
  rindexList *curInd;
  unsigned short int nrDiffs = 0;
  rp_cur = head;

  if ((diffHead = mk_diffList_post ()) == NULL)
    fprintf (stderr, "Allocation of memory for a new diffListpost failed!\n");

  if ((curDiff = mk_diffList_post ()) == NULL)
    fprintf (stderr, "Allocation of memory for a new diffListpost failed!\n");

  diffHead->next = NULL;
  diffHead->ecode = NULL;
  diffHead->time = NULL;
  diffHead->date = NULL;
  diffHead->rindexList = NULL;

  if (rp_cur != NULL)
  {
    do
    {
      if (rp_cur->mysqlPost != NULL)
      {
        diffHead = addDiff (diffHead, rp_cur->mysqlPost);
      }
    }
    while ((rp_cur = rp_cur->next_post) != NULL);
  }
  else
    printf ("An empty list\n");
  /*
   * Evaluating results ***********************
   */
  nrDiffs = printRes (diffHead, minDiff);
  /*********************************************/
  /*
   * Return memory
   */
  while (diffHead->next != NULL)
  {
    curDiff = diffHead->next;
    diffHead->next = curDiff->next;
    while (curDiff->rindexList->next != NULL)
    {
      curInd = curDiff->rindexList;
      curDiff->rindexList = curInd->next;
      free (curInd);
    }
    free (curDiff->rindexList);
    free (curDiff);
  }
  free (diffHead);
  return nrDiffs;
}

/*
 * Displays diffList AND saves the last entry that is <= minDiff.
 * Returns: the number of diffusions.
 */
int printRes (
  diffList * diffHead,
  unsigned int *minDiff)
{
  unsigned short int nrRouterErr = 0;
  diffList *cur;
  rindexList *curIndex;
  boolean DoubleReport = false;
  unsigned short int nrDiffs = 0;
  cur = diffHead;

  /*
   * Hack.
   * * If the first entry is equal to the last in the previuos window,
   * * it will will be discarded, not reported twice.
   */
  if ((strcmp (cur->time, dp->time) == 0)
      && (strcmp (cur->date, dp->date) == 0)
      && (strcmp (cur->ecode, dp->ecode) == 0))
  {
    DoubleReport = true;
  }

  while (cur != NULL)
  {
    curIndex = cur->rindexList;
    /*
     * Count routers in list
     */
    while (curIndex != NULL)
    {
      nrRouterErr += 1;
      curIndex = curIndex->next;
    }
    if ((nrRouterErr >= *minDiff) && (DoubleReport == false))
    {
      nrDiffs += 1;
      printf ("  _____________________________________________________\n");
      printf (" |                   Diffusion                        |\n");
      printf (" |----------------------------------------------------|\n");
      cur->date += 2;           /* Skip '20' in '2003' */
      if (strcmp (cur->ecode, "") == 0)
        printf (" |     Error: Best guess is that this is a Traceback   |\n");
      else
        printf (" |     Error: %-41s|\n", cur->ecode);
      printf (" |     Diffusion error first at %s %-14s|\n", cur->time,
              cur->date);
      printf (" |     Number of routers in list %-22d|\n", nrRouterErr);
      printf (" |____________________________________________________|\n");

      /*
       * Since we dont know which one that will be last we must do this:
       */
      strcpy (dp->date, "20");
      strcpy (dp->ecode, cur->ecode);
      strcpy (dp->time, cur->time);
      strcat (dp->date, cur->date);

    }
    nrRouterErr = 0;

    cur = cur->next;
    DoubleReport = false;       /* Only once per time-window */
    curIndex = NULL;
  }
  return nrDiffs;
}

/*
```

```
 * Adds an error to diffList if the error is not
 * already in diffList. diffList's rindexlist will be updated
 * with the router reporting the error, but only if the router doesn't
 * already exist in diffList->rindexlist.
 *
 * Returns: Head of diffList
 */
diffList *addDiff (
  diffList * diffHead,
  MYSQL_POST * mp)
{
  boolean inList = false;
  diffList *cur;
  diffList *newpost;
  cur = diffHead;

  while (cur != NULL)
  {
    if (cur->ecode != NULL)
    {
      if ((strcmp (cur->ecode, mp->ecode)) == 0)
      {
        inList = true;
      }
    }
    if (inList || (cur->next == NULL))
      break;

    cur = cur->next;
  }

  if (!inList)
  {
    if ((newpost = mk_diffList_post ()) == NULL)
      fprintf (stderr, "Alloc. of memory for new diffList-post failed!\n");

    newpost->ecode = mp->ecode; /*ref to error    */
    newpost->time = mp->time;   /*ref first seen  */
    newpost->date = mp->date;   /*ref first seen  */
    newpost->next = NULL;
    newpost->rindexList = NULL;
    newpost = addDiffRt (newpost, mp->rindex);  /* Add router if not in list */

    if (diffHead->ecode == NULL)
    {                           /* Its the first post */
      diffHead = newpost;
    }
    else
      cur->next = newpost;
  }
  else
  {                            /* Error already saved,  update router list for this error */
    cur = addDiffRt (cur, mp->rindex);  /* Add router to error list */
  }
  return diffHead;
}

/*
 * Adds a router to diffList's rindexList if
 * rindex doesn't exist in rindexList.
 *
 * Returns: dcur
 */
diffList *addDiffRt (
  diffList * dcur,
  int *rindex)
{
  boolean inList = false;
  rindexList *newpost;
  rindexList *cur;
  cur = dcur->rindexList;

  while (cur != NULL)
  {
    if (*cur->rindex == *rindex)
    {
      inList = true;
    }
    if (inList || cur->next == NULL)
      break;
    cur = cur->next;
  }
  if (!inList)
  {
    if ((newpost = mk_rindexList_post ()) == NULL)
      fprintf (stderr, "Allocation of memory for new rindexpost failed!\n");

    newpost->next = NULL;
    newpost->rindex = rindex;

    if (cur == NULL)
    {
      dcur->rindexList = newpost;
    }
    else
    {
      cur->next = newpost;
    }
  }
  //else ignore this router, already in list.

  return dcur;
}

/*
 * Retrieves data from MySQL databas
 * sqlquery is a complete sql query
 * toffset is a global reference to the list containing
 * retrieved data from the MySQL database.[listCommon.h]
 * Returns :
 * Non-local (MYSQL_POST*) head with the retrived mysql data
 */
MYSQL_POST *getList (
  char *sqlquery)
{
  MYSQL_POST *current = NULL;
  MYSQL_RES *result;
  MYSQL_ROW row;
  unsigned int num_fields = 0;
  unsigned int num_rows = 0;

  head = mk_MYSQL_post (NULL);  /* Create the list */
  current = head;
  toffset = head;               /* Table offset non-local */

  if (mysql_query (&connection, sqlquery))
    MYSQL_exiterr (1);

  if (!(result = mysql_use_result (&connection)))
    MYSQL_exiterr (2);

  num_fields = mysql_num_fields (result);

  if (num_fields > 0)
  {
    row = mysql_fetch_row (result);
    set_tb_val (current, row);

    while ((row = mysql_fetch_row (result)))
    {
      current = mk_MYSQL_post (current);
      set_tb_val (current, row);
    }
  }
  else
  {
    printf ("No hits on your query, please try another time period \n");
  }
  /*
   * mysql_num_rows() will not return the correct value until all
   * * the rows in the result set have been retrieved.
   * * This is why it's placed here
   */
  num_rows = mysql_num_rows (result);
  //printf("<--- Nr of hits in db = %d --->\n",num_rows );

  if (mysql_errno (&connection))        // mysql_fetch_row() failed due to an error
    MYSQL_exiterr (3);

  mysql_free_result (result);
  mysql_close (&connection);

  return head;
}

/*
 * Saves fetched mysql row (MYSQL_ROW) into (MYSQL_POST*) cur
 */
void set_tb_val (
  MYSQL_POST * cur,
  MYSQL_ROW r)
{

  if (sizeof (r) != 4)
  {
    printf ("Invalid sql response\n");
    terminator ();
  }
  if (cur == NULL)
  {
    printf ("List not created or defect (null)\n");
    terminator ();
  }
  cur->rindex = (int *) malloc (sizeof (int));
  *cur->rindex = atoi (r[0]);
  cur->time = (char *) strdup (r[1]);
  cur->date = (char *) strdup (r[2]);
  cur->ecode = (char *) strdup (r[3]);
}

/*
 * Allocate memory for dp non-local pointer.
 */
diffList *initHackPointer (
  )
{
  dp = mk_diffList_post ();
  dp->ecode = (char *) calloc (25, 1);
  dp->time = (char *) calloc (25, 1);
  dp->date = (char *) calloc (25, 1);
  return dp;
}
```

A.2 diffusion.h

```
/***************************************************************************
 *                       diffusion.h  -  Headerfile                       *
 *                          -------------------                           *
 *  begin               : Thu Sep 05 2002                                 *
 *  copyright           : (C) 2002 by Henrik Skantz & Mikael BÃũhm         *
 *                            Bachelors project - KaU fall 2002           *
 *  email               : hesk@linux.se, micko@linuxmail.org              *
 ***************************************************************************/
/***************************************************************************
 *   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 of the License, or     *
 *   (at your option) any later version.                                   *
 ***************************************************************************/
#ifndef DIFFUSION_FILE
#define DIFFUSION_FILE
/****************************************************************************/
/*
 * Constants
 */
/****************************************************************************/
/****************************************************************************/
/*
 * Non local variables
 */
/****************************************************************************/
int logerr;

/****************************************************************************/
/*
 * Type definitions
 */
/****************************************************************************/
#define DEBUG       0           /* Set true to get degubbing info */
//#define ROUTSIZE  100 /* Max number of routers in network */
#define USAGEMESS "Usage: diffusion <start-day [2003-03-21]> \
<end-day [2003-03-31]>  <windowsize [seconds]> \
<min routers [2 - max availible in network]> <system[cisco]\n"
```

```
#define LOCKFILE "/tmp/mylogd-lock"
#define LF_CLOSE_TEXT "Lockfile not REMOVED, if you started mylogd with the \
-file flag nothing is wrong.\n Else be sure to remove it before starting \
mylogd again\n"

#define LF_ERROR_TEXT "Lockfile exists, mylogd already running, if not \
remove lockfile /tmp/mylogd-lock"
#define MYLOGDFILE "/var/log/mylogd/mylogd.log"
#define MYLOGDMODE "a+"

/***************************************************************************/
/*
 * Function prototypes
 */
/***************************************************************************/
void diffusion (
  char **in);
MYSQL_POST *getList (
  );
void set_tb_val (
  MYSQL_POST * cur,
  MYSQL_ROW r);
void display_summary (
  char **in,
  unsigned int winSize,
  unsigned int minDiff,
  unsigned short int nrDiffs);
int findDiff (
  winError * rp_cur,
  unsigned int *minDiff);
diffList *addDiff (
  diffList * diffHead,
  MYSQL_POST * mp);
diffList *addDiffRt (
  diffList * cur,
  int *rindex);

int printRes (
  diffList * diffHead,
  unsigned int *minDiff);
int sumErrors (
  int winSize,
  winError * rp_cur);
int chkargs (
  char **inargs);

/*
 * Hack. Previous time-window entry
 */
```

```
diffList *dp;
diffList *initHackPointer (
  );

#endif
```

Appendix B

Recurring sequence files

B.1 rekerr.c

```
/****************************************************************************
                           rekerr.c  -  description
          ----<<0>>--- * Find recurring errors *  ---<<0>>---
                             -------------------
    begin                : Wen Apr 23 12:00:00 CET 2003
    copyright            : (C) 2003 by Mikael Bï£¡m & Henrik Skantz
    email                : micko@linuxmail.org, hesk@linux.se
*****************************************************************************/
/****************************************************************************
 *   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 of the License, or     *
 *   (at your option) any later version.                                   *
 ***************************************************************************/
#include "header.h"

int main (
  int argc,
  char **argv)
{

  if (argc != 6 && argc != 7)
  {
    printf (USAGEREK);
    exit (FAIL);
  }
  else
    rekerr (argv);

  return 0;
}
```

```
/*
 * Collect uniq sequences of repeting router/errors
 */
void rekerr (
  char **in)
{

  Config *C1 = (Config *) malloc (sizeof (Config));

  errList *eListhead = NULL;    /* data from database */
  SeqList *seqListhead = NULL;  /* List for found patterns  */

  unsigned int seqrep = 0;      /* Minimum nr of sequence repetitions */
  unsigned int seqlen = 0;      /* Minimum sequence length */

  char *show_all = NULL;

  /*
   * Read the mylogd config file
   */
  readConf (CONFFILE, C1);      /* Read config to C1 */

  seqlen = (unsigned) atoi (in[3]);
  seqrep = (unsigned) atoi (in[4]);

  if (in[6] != NULL)
  {
    show_all = in[6];
  }

  if (seqlen < MINPLEN)
    seqlen = MINPLEN;

  /*
   * Set time from inargs
   */
  initTimeObjs (in[1], in[2]);

  /*** GET DATA FROM DATABASE ***********************************************/

  /*
   * Connect to database
   */
  if (connectDB (C1) == NULL)
  {
    fprintf (stderr, SQLCONN);
    free (C1);
    free (eListhead);
    terminator ();
  }
  while (mktime (fr_) <= mktime (to_))
  {
    eListhead = collect_db_data (in, C1, eListhead);
  }
  /*** PROCESS THE DATA *****************************************************/
  /*
   * Allocate memory for Lists and variables
   */

  seqListhead = (SeqList *) malloc (sizeof (SeqList));
  seqListhead->next = NULL;
  seqListhead->posList = NULL;
  /*
   * run alg. for pattern detection
   */
  seqListhead = findSeq (seqListhead, eListhead, seqlen);

  /*** WRITE REPORT *********************************************************/
  print_SeqList (seqListhead, seqrep, show_all);
  printf (SUMPTRN, seqlen);
  printf (SUMREP, seqrep);

  /*** FREE ALL MEMORY *****************************************************/
  free (to_);
  free (fr_);
  free (st_);
  free (C1);

  if (rmerrList (eListhead) != NULL)
    printf (ELISTERR);
}

/*
 * Collect data from database, returns eListhead which is a ref. to
 * ** this data.
 */
errList *collect_db_data (
  char **in,
  Config * C1,
  errList * eListhead)
{

  char *dbtable = NULL;
  char *sqlquery = NULL;

  /*
   * List containing result of mysql query
 */
MYSQL_POST *logList = NULL;

/*
 * Set dbtable from inargument in[1]
 */
if ((dbtable = setdbtable (in[1], in[5])) == NULL)
{
  fprintf (stderr, DBTBLERR);
}

if (existTable (dbtable))
{
  /*
   * Create an sqlquery for data:[rindex, ltime, localtime, errorcode]
   */
  if ((sqlquery = rekSqlQuery (dbtable)) == NULL)
  {
    fprintf (stderr, SQLQRERR);
  }
  /*
   * Mysql get complete interval for valid table
   */
  printf (SQLCOL);
  logList = getPosts (sqlquery);
  printf (SQLQRDONE);

  /*
   * Store uniq errors in a list
   */
  eListhead = storeErr (logList, eListhead);

  /*
   * Done with logList, make it empty
   */
  logList = rm_SQL_List (logList);
}
else
{
  printf (DBTBNF, dbtable);
  terminator ();
}

/*
 * Move forward to next month
 */
incr_month (1);
*in[1] = set_tablemon (in[1]);
```

```
  /*
   * Connect to database
   */
  if (connectDB (C1) == NULL)
  {
    fprintf (stderr, SQLRECON);

    if (rmerrList (eListhead) != NULL)
      printf (ELISTERR);

    free (C1);
    free (eListhead);
    terminator ();
  }

  free (dbtable);
  free (sqlquery);

  return eListhead;
}

/*
 * search for patterns (of lenght seqlen) in eListHead based on
 * ** router id and error . seqListhead is return which contains a list
 * ** with each node containing a list for a certain pattern.
 * ** Every node in the seqListhead list contains a number that represent
 * ** the number of times it was found, at least the 'seqlen' first .
 */
SeqList *findSeq (
  SeqList * seqListHead,
  errList * eListhead,
  unsigned seqlen)
{
  errList *startSeq = NULL;     /* Ref. to sequence to search */
  errList *searchSeq = NULL;    /* Ref. to sequence to compare */
  errList *curSeq = NULL;
  errList *curSearchSeq = NULL;
  errList *FirstInStart = NULL; /* Ref. to start sequence  */
  PosList *PtrnPosList = NULL;  /* List containing a found pattern */
  SeqList *seqList = NULL;      /* List containing all patterns */
  boolean ptrnFound = false;    /* Whether or not pattern is found */
  unsigned int patternLen = 0;  /* Length of found pattern  */
  unsigned int cur_seqrep = 0;  /* No. of recurrence for a pattern  */
  unsigned int stp = 0;         /* steps to next pattern to search for */
  unsigned int maxLen = seqlen;
  int seqExistLen = 0;
  startSeq = eListhead;
  searchSeq = eListhead;
  seqList = seqListHead;
```

```
/*
 * printf("%d %s %s\n",*eListhead->rindex,eListhead->ecode,eListhead->ttime);
 */
printf (REKSTART);

/*
 * Set searchSeq which is to be compared with startSeq
 */
searchSeq = setSearchSeqPos (startSeq, searchSeq, seqlen);

/*
 * Search until end of eListhead, which is when searchSeq == NULL
 */
while (true)
{
  /*
   * Look only for new patterns
   */
  if ((seqExistLen = seqExist (seqList, startSeq)) == -1)
  {
    FirstInStart = startSeq;

    while (searchSeq != NULL)
    {
      /*
       * set position for searchSeq to start next time
       */
      curSearchSeq = searchSeq->next;

      /*
       * Compare first node
       */
      if (comparePost (startSeq, searchSeq))
      {
        curSeq = startSeq->next;
        patternLen = 1;
        /*
         * Compare each post until no more match.
         * * This can be longer than seqlen
         */
        while ((comparePost (curSeq, curSearchSeq) == true) &&
               (comparePost (curSeq, FirstInStart) == false))
        {
          curSeq = curSeq->next;
          curSearchSeq = curSearchSeq->next;
          patternLen += 1;
        }
        if (patternLen >= seqlen)
```

```
          {
            /*
             * Save pos for pattern
             */
            ptrnFound = true;
            PtrnPosList = PtrnIns (PtrnPosList, searchSeq, patternLen, 0);
            cur_seqrep += 1;
            /*
             * Set max length of seq in pattern
             */
            if (patternLen > maxLen)
              maxLen = patternLen;
          }
        }
        /*
         * Set new search position
         */
        searchSeq = curSearchSeq;
      }                         /* End While(searchSeq != NULL) */

      /*
       * when search for startSeq in searchseq is complete add patternlist
       * * to main list. Then move startSeq to next sequence to be found.
       */
      if (ptrnFound == true)
      {
        /*
         * The first pattern(startSeq) is a found pattern
         */
        /*
         * PtrnIns adds at the beginning of list ->
         * ** startSeq comes first in list (PtrnPosList)
         */
        PtrnPosList = PtrnIns (PtrnPosList, startSeq, seqlen, cur_seqrep + 1);
        /*
         * Insert new patternlist to mainlist
         */
        seqList = seqIns (seqList, PtrnPosList);
        PtrnPosList = NULL;
        ptrnFound = false;
        /*
         * Search for completly new pattern. curSeq > one step
         */
        //startSeq = curSeq; /* Move startSeq to next sequence */
        /*
         * Move forward max length of found sequence to aviod subsequences
         */
        for (stp = 0; stp < maxLen; stp++)
          startSeq = startSeq->next;
        cur_seqrep = 0;
      }
      /*
       * Move startSeq only one step:
       * ** if pattern was already saved or searchSeq at end (null)
       */
      else
      {
        startSeq = startSeq->next;      /* Move startSeq to next sequence */
      }
    }      /*End if(seqExist(seqList, startSeq) == false)  */
    /*
     * Sequence already matched as a pattern (in seqList)
     */
    /*
     * Do not search after subsequences in already searched sequences
     */
    else
    {
      //  printf("Hoppar %d steg\n",seqExistLen);
      for (stp = 0; stp < (unsigned) seqExistLen; stp++)
{/* Move startSeq to next sequence, no subsequence */
        startSeq = startSeq->next;
}
    }
    /*
     * Set a new SearchSeq position, if NULL search is complete!
     */
    if ((searchSeq = setSearchSeqPos (startSeq, searchSeq, seqlen)) == NULL)
      break;
  }
  /*
   * Return the head of list containing
   * ** repeted errors in different patterns
   */
  return seqList;
}

/*
 * Prints the (first, if show_all != show_all, else all)
 * **  PosList node in every SeqList node.
 * ** (the pattern)
 * ** And the number of times the pattern was found (SeqList member rep).
 */
void print_SeqList (
  SeqList * cur,
  unsigned seqrep,
  char *show_all)
{
```

```
SeqList *Seqtmp;
PosList *Postmp;
boolean sa = false;
int RepCount = 0;
unsigned int SeqLen = 0;
boolean longPtrn = false;     /* Longeset pattern indicator */
unsigned int SeqReps = 0;     /* Repetitions for the longest pattern */
int ptrLen[600];              /* Holds the rep. for the longest patterns */
int ptrC = 0;
Seqtmp = cur;

if (show_all != NULL)
  if (strcmp (show_all, "show_all") == 0)
    sa = true;

while (Seqtmp != NULL)
{
  Postmp = Seqtmp->posList;
  SeqReps = 0;
  SeqLen = 0;
  if (Postmp != NULL)
  {
    if (Postmp->rep >= seqrep)
    {
      RepCount++;
      printf (PRINTSUM, RepCount, Postmp->rep);
      if (sa == true)
      {                       /*in show_all patterns */
        printf (" This is the first occurrence of this pattern.\n \
        - The rest are listed in reversed order -\n");
      }
      print_PosNode (Postmp); /*Always print the first pattern */
      //if(sa == true){ /*in show_all patterns*/
      //printf(DELIM);
      while (Postmp != NULL)
      {
        /*
         * Set max length of a pattern
         */
        if (SeqLen < Postmp->len)
        {
          longPtrn = true;
          SeqLen = Postmp->len;
          SeqReps = 0;
        }
        /*
         * Count repetitions for the longest pattern
         */
```

```
          if (SeqLen == Postmp->len)
            SeqReps += 1;

          if (sa == true)
          {                    /*in show_all patterns */
            printf (DELIM);
            print_PosNode (Postmp);
          }
          //printf(DELIM);
          Postmp = Postmp->next;
        }
        /*
         * If the longest pattern was rep. enough times, save it.
         */
        if (SeqReps >= seqrep)
        {
          if (ptrC < 500)
          {
            ptrLen[ptrC] = SeqLen;
            ptrC += 1;
          }
        }
        //}
      }
    }
    Seqtmp = Seqtmp->next;
  }
  /*
   * Print small summary
   */
  printf (DELIM);
  printf ("\n -Summary- \n");
  SeqLen = MaxLen (ptrLen, ptrC);

  printf ("The longest pattern was %d routers long, repeated %d times\n",
          SeqLen, seqrep);

}

int MaxLen (
  int l[],
  int ptrC)
{
  int k;
  int max = 0;
  for (k = 0; k < ptrC; k++)
  {
    if (max < l[k])
      max = l[k];
  }
  return max;
}

/*
 * Prints all elements in a PosList node
 */
void print_PosNode (
  PosList * pNode)
{

  errList *seq;
  unsigned short int i;
  unsigned short int k;

  k = pNode->len;
  seq = pNode->seqStart;

  for (i = 0; i < k; i++)
  {
    printf (PRINTPOS,
            *seq->rindex, seq->rname, seq->ecode, seq->ttime, seq->date);
    seq = seq->next;
  }
}

/*
 * Allocates and returns memory for a PosList
 */
PosList *mk_PosNode (
  )
{

  PosList *new_post;

  if ((new_post = (PosList *) malloc (sizeof (PosList))) == NULL)
    fprintf (stderr, MEMERR);

  new_post->next = NULL;

  return new_post;

}

/*
 * Adds an errList at beginning of PosList.
 * ** The new PosList is returned
 */
PosList *PtrnIns (
  PosList * PtrnPosHead,
  errList * curPos,
  int ptrnLen,
  unsigned seqrep)
{
  PosList *PtrnCur = NULL;

  PtrnCur = mk_PosNode ();
  PtrnCur->next = PtrnPosHead;

  PtrnCur->seqStart = curPos;
  PtrnCur->len = ptrnLen;
  PtrnCur->rep = seqrep;

  return PtrnCur;               //New head
}

/*
 * Allocates and returns memory for a SeqList
 */
SeqList *mk_SeqNode (
  )
{

  SeqList *new_post;

  if ((new_post = (SeqList *) malloc (sizeof (SeqList))) == NULL)
    fprintf (stderr, MEMERR);

  new_post->next = NULL;

  return new_post;

}

/*
 * Insert PtrnPosList at first position in seqlist
 * ** The new seqlist is returned
 */
SeqList *seqIns (
  SeqList * seqlist,
  PosList * PtrnPosList)
{

  SeqList *cur = NULL;

  cur = mk_SeqNode ();
  cur->next = seqlist;
  cur->posList = PtrnPosList;

  return cur;                  //New Head
}

/*
 * Check if StartSeq is referenced in seqList->seqStart
 * ** Returns the number of entries in the seq.
 * ** if StartSeq is referenced in SeqList
 * ** otherwise -1
 */
int seqExist (
  SeqList * seqList,
  errList * StartSeq)
{

  SeqList *cur;
  PosList *poscur;

  cur = seqList;

  while (cur != NULL)
  {
    poscur = cur->posList;
    while (poscur != NULL)
    {
      if (poscur->seqStart->ecode == StartSeq->ecode)
        return poscur->len;
      poscur = poscur->next;
    }
    cur = cur->next;
  }
  return -1;
}

/*
 * Sets  searchSeq to point at startSeq + seqlen.
 * ** Reference to searchSeq is returned
 */
errList *setSearchSeqPos (
  errList * startSeq,
  errList * searchSeq,
  unsigned seqlen)
{
  unsigned short i;

  searchSeq = startSeq;

  if (searchSeq == NULL)
```

```
    return NULL;

  for (i = 0; i <= seqlen; i++)
    if ((searchSeq = searchSeq->next) == NULL)
      break;

  return searchSeq;
}

/*
 * Compares two post, if they are equal true is returned
 * ** else false
 */
boolean comparePost (
  errList * startSeq,
  errList * searchSeq)
{

  if (startSeq == NULL || searchSeq == NULL)
    return false;

  if (strcmp (startSeq->ecode, searchSeq->ecode) == 0)
    if (strcmp (startSeq->rname, searchSeq->rname) == 0)
      return true;

  return false;
}

/*
 * Returns number of nodes in an errList
 */
int countListLen (
  errList * eListhead)
{

  int len = 0;
  errList *eListcur;

  eListcur = eListhead;

  if (eListhead == NULL)
    return 0;

  while (eListcur->next != NULL)
  {
    len++;
    eListcur = eListcur->next;
  }
  return len;
}

/*
 * Remove the list(eListhead) and returns the head of an empty list
 */
errList *rmerrList (
  errList * head)
{

  errList *cur;

  while (head->next != NULL)
  {
    cur = head->next;
    head->next = cur->next;
    free (cur);
  }
  free (head);

  return (head = NULL);
}

/*
 * Save errors uniq with statistic data from SQL-query
 */
/*
 * List from MYSQL-query must be in accending order of time
 */
errList *storeErr (
  MYSQL_POST * logList,
  errList * head)
{

  errList *cur, *newpost;

  cur = head;

  if (cur != NULL)
    /*
     * Roll to end of list :-p NOT EFFECTIVE AT ALL IF MANY CALLS
     */
    while (cur->next != NULL)
      cur = cur->next;


  newpost = (errList *) malloc (sizeof (errList));
  newpost->next = NULL;
  /*
```

```
   * If list is empty dont miss filling data to the first one
   */
  if (head == NULL)
  {
    head = newpost;
    cur = head;
  }
  while (logList != NULL)
  {
    newpost->ecode = logList->ecode;
    newpost->rindex = logList->rindex;
    newpost->rname = logList->rname;
    newpost->ttime = logList->time;
    newpost->date = logList->date;
    newpost->time = newpost->time = getUTC (logList);

    cur->next = newpost;
    cur = cur->next;

    if (logList->next_post == NULL)
      break;

    logList = logList->next_post;
    /*
     * Allocate memory for a new errorList post
     */
    newpost = (errList *) malloc (sizeof (errList));
    newpost->next = NULL;
  }

  return head;
}
```

B.2 rekerr.h

```
/***************************************************************************
 *                        rekerr.h  -  Headerfile                          *
 *                          -------------------                            *
 *  begin               : Wen Apr 23 2003                                  *
 *  copyright           : (C) 2003 by Henrik Skantz & Mikael Böhm          *
 *                            D-Level Reserch                              *
 *  email               : hesk@linux.se, micko@linuxmail.org               *
 ***************************************************************************/
/***************************************************************************
 *   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 of the License, or     *
 *   (at your option) any later version.                                   *
 ****************************************************************************/
#ifndef REKERR_FILE
#define REKERR_FILE

/****************************************************************************/
/*
 * Non local variables
 */
/****************************************************************************/
struct tm *to_;        /* Time struct, keeps track of "to" date and time   */
struct tm *fr_;        /* Time struct, keeps track of "from" date and time */
struct tm *st_;        /* Time struct for "starttime", keep this static    */
unsigned int nr_set;   /* Number of rows on each query on database         */
time_t interval;       /* Total time interval of search                    */

/****************************************************************************/
/*
 * Constants
 */
/****************************************************************************/
#define MINPLEN       3
#define USAGEREK "Usage: rekerr <start-day [2003-03-21]> \
<end-day [2003-05-01]> <min errors [3 - num routers in network] in pattern> \
<sequence repetitions (>2) > <system[cisco]> [show_all * Optional *]\n"
#define PRINTSUM "\n Pattern [%d] beginning with this sequence was \
repeated %d times \n"
#define DELIM "-----------------------------------------------------------\n"
#define PRINTPOS "Router:(%d) %-20s  %-30s [%s | %s]\n"
#define MEMERR  "Allocation of memory for new post failed!\n"
#define REKSTART "Searching for patterns...(This can take quite some time)\n"
#define DBTBLERR  "Could not set dbtable!\n"
#define DBTBNF "Table %s doesn't exist\n"
#define SQLQRERR "Could not create SQL-query for this table!\n"
#define SQLCOL "Collecting data from MySQL server...\n"
#define SQLQRDONE "Done collecting\n"
#define SQLRECON "Re-connection to database failed!\n"
#define SQLCONN  "Connection to database failed!\n"
#define ELISTERR "List [errorList] was NOT removed from mem\n"
#define SUMPTRN "Shortest pattern to search           = %d\n"
#define SUMREP "Minimum repetition for pattern       = %d\n"
#define NODAY "No day interval \n"
/****************************************************************************/
/*
 * Type definitions
 */
/****************************************************************************/
/*
 * Holds a list of all logs to evaluate
 */
```

```
struct errList
{
  char *ecode;                /* Error                              */
  int *rindex;                /* Router  (id)                       */
  char *rname;                /* Router Name                        */
  time_t time;                /* Time found  (sec)                  */
  char *date;                 /* date                               */
  char *ttime;                /* readable time                      */
  errList *next;              /* Next post in list                  */
  boolean refIndicator;       /* Indicates if post is ref. by rekList  */
};

/*
 * List of repeting sequences
 */
struct SeqList
{
  PosList *posList;
  SeqList *next;
};

/*
 * Holds start and stop positions for sequences
 */
struct PosList
{
  unsigned int len;
  unsigned int rep;
  errList *seqStart;
  PosList *next;
};

/****************************************************************************/
/*
 * Function prototypes
 */
/****************************************************************************/
void rekerr (
  char **in);

errList *storeErr (
  MYSQL_POST * logList,
  errList * head);
errList *rmerrList (
  errList * head);

int MaxLen (
  int l[],
  int ptrC);
int countListLen (
  errList * head);
SeqList *findSeq (
  SeqList * seqListHead,
  errList * eListhead,
  unsigned int seqlen);

errList *setSearchSeqPos (
  errList * startSeq,
  errList * searchSeq,
  unsigned seqlen);

boolean comparePost (
  errList * startSeq,
  errList * searchSeq);
int seqExist (
  SeqList * seqList,
  errList * StartSeq);

PosList *PtrnIns (
  PosList * p,
  errList * searchSeq,
  int ptrnLen,
  unsigned seqrep);
SeqList *seqIns (
  SeqList * seqlist,
  PosList * PtrnPosList);


PosList *mk_PosNode (
  );
SeqList *mk_SeqNode (
  );

void print_PosNode (
  PosList * pNode);
void print_SeqList (
  SeqList * cur,
  unsigned seqrep,
  char *show_all);

errList *collect_db_data (
  char **in,
  Config * C1,
  errList * eListhead);
#endif
```

B.3 rekerrMySQL.c

```
#include "header.h"

/*
 * Saves MYSQL_ROW into MYSQL_POST
 * Returns: MYSQL_POST
 */
MYSQL_POST *setPostValue (
  MYSQL_POST * cur,
  MYSQL_ROW row)
{

  if ((sizeof (row) != 4) || row == NULL)
  {
    printf ("Invalid sql response or empty set on query\n");
    exit (-1);
  }
  if (cur == NULL)
  {
    printf ("List not created or defect (null)\n");
    exit (-1);
  }
  cur->rindex = (int *) malloc (sizeof (int));
  *cur->rindex = atoi (row[0]);
  cur->rname = (char *) strdup (row[1]);
  cur->time = (char *) strdup (row[2]);
  cur->date = (char *) strdup (row[3]);
  cur->ecode = (char *) strdup (row[4]);

  return cur;
}

/*
 * Get data from MySQL database
 * Non-L-E (MYSQL_POST*) head is set to the retrived  mysql data.
 * Returns: head of  MYSQL_POST* containing MySQL data
 */
MYSQL_POST *getPosts (
  char *sqlquery)
{

  MYSQL_POST *current = NULL;
  MYSQL_RES *result;
  MYSQL_ROW row;
  unsigned int num_fields = 0;
  unsigned int num_rows = 0;

  head = mk_MYSQL_post (NULL);  /* Create the list */
```

```
current = head;

/****start debug ***********************************/
//  printf ("SQL=%s\n", sqlquery);
//strcat(sqlquery," limit 10");
printf ("SQL=%s\n", sqlquery);
/****end debug *************************************/

if (mysql_query (&connection, sqlquery))
  MYSQL_exiterr (1);

if (!(result = mysql_use_result (&connection)))
  MYSQL_exiterr (2);

num_fields = mysql_num_fields (result);

if (num_fields > 0)
{
  /*
   * (no extra post at end of list)
   */
  row = mysql_fetch_row (result);
  current = setPostValue (current, row);

  while ((row = mysql_fetch_row (result)))
  {
    current = mk_MYSQL_post (current);
    current = setPostValue (current, row);
  }
}
else
{
  printf ("No hits on your query, please try another time period \n");
}
/*
 * mysql_num_rows() will not return the correct value until all
 * * the rows in the result set have been retrieved.
 * * This is why mysql_num_rows() is placed here
 */
nr_set = num_rows = mysql_num_rows (result);
printf ("<--- Nr of hits in db = %d --->\n", num_rows);

if (mysql_errno (&connection))// mysql_fetch_row() failed due to an error
  MYSQL_exiterr (3);

mysql_free_result (result);
mysql_close (&connection);

return head;
```

```
}
```

B.4 rekerrMySQL.h

```
/***************************************************************************
 *                        rekerr.h  -  Headerfile                          *
 *                          -------------------                            *
 *  begin                : Wen Apr 23 2003                                 *
 *  copyright            : (C) 2003 by Henrik Skantz & Mikael Böhm         *
 *                             D-Level Reserch                             *
 *  email                : hesk@linux.se, micko@linuxmail.org              *
 ***************************************************************************/
/***************************************************************************
 *   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 of the License, or     *
 *   (at your option) any later version.                                   *
 ***************************************************************************/
#ifndef REKERR_MYSQL_FILE
#define REKERR_MYSQL_FILE

MYSQL_POST *getPosts (
  char *sqlquery);
MYSQL_POST *setPostValue (
  MYSQL_POST * cur,
  MYSQL_ROW row);


#endif
```

Appendix C

Low frequent files

C.1 lowfreq.c

```
/***************************************************************************
                          lowfreq.c  -  description
        ----<<0>>--- * Find low frequency errors *  ---<<0>>---
                             -------------------
    begin                : Mon Apr 09 11:00:00 CET 2003
    copyright            : (C) 2003 by Mikael Böhm & Henrik Skantz
    email                : micko@linuxmail.org, hesk@linux.se
 ***************************************************************************/
/***************************************************************************
 *   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 of the License, or     *
 *   (at your option) any later version.                                   *
 ***************************************************************************/
#include "header.h"

int main (
  int argc,
  char **argv)
{

  /*
   * argv -<<>>- Start/slut datum, min antal fel, systemtyp
   */
  if (argc != 6)
  {
    printf (USAGELOW);
    exit (FAIL);
  }
  else
    lowfreq (argv);
  return 0;
}

/*
 * Collect errors uniq and count them to find lowfreqvency errors
 */
void lowfreq (
  char **in)
{
  char *fromdate = NULL;
  unsigned int numberr = atoi (in[3]);
  unsigned int maxgrad = atoi (in[4]);
  unsigned int errorcounter = 0;
  unsigned int erlimit = 0;
  Config *C1 = (Config *) malloc (sizeof (Config));

  /*
   * List of uniq errors to count
   */
  errorList *eListhead = NULL;

  /*
   * Allocate memory for errorList with uniq errors
   */
  if ((eListhead = (errorList *) malloc (sizeof (errorList))) == NULL)
  {
    fprintf (stderr, "Allocation of memory for errorList failed!\n");
    terminator ();
  }

  /*
   * Make sure list is empty
   */
  eListhead->next = NULL;

  /*
   * Read the mylogd config file
   */
  readConf (CONFFILE, C1);      /* Read config to C1 */

  /*
   * Set a static from date to sql-query
   */
  fromdate = getArrayPos (in, 1);

  /*
   * Set time period objekts
   */
  setTimePeriod (in[1], in[2]);

  /*
   * Connect to database
   */
  connectToDB (C1);

  /*
   * Collect all errors in time period
   */
  eListhead = getErrors (in, fromdate, eListhead, numberr, C1);

  /*
   * Count the number of errors in list
   */
  errorcounter = countErrors (eListhead);

  /*
   * Set upper limit to display for total set
   */
  erlimit = setLFErrorLim (LFRQFACT, errorcounter, numberr);

  /*
   * Evaluate errors and print report
   */
  printLFReport (eListhead, erlimit, maxgrad, errorcounter, fromdate, in);

  /*
   * Time to set all memory free
   */
  free (to_);
  free (fr_);
  free (st_);
  free (C1);
  free (fromdate);

  /*
   * Remove all entries in the errorlist
   */
  if (rmeList (eListhead) == NULL)
    printf ("List [errorList] is removed from mem\n");
}

/*
 * Allocates new memory for array at pos and return as char
 */
char *getArrayPos (
  char **in,
  int pos)
```

```
{
  char *str = NULL;
  str = (char *) malloc (sizeof (in[pos]));
  return strcpy (str, in[pos]);
}

/*
 * Setting the global time objekts values
 */
void setTimePeriod (
  char *fr,
  char *to)
{
  /*
   * Set time from inargs
   */
  initTimeObjs (fr, to);
  interval = mktime (to_) - mktime (fr_);

  if (interval < 0)
  {
    fprintf (stderr, "Lowfreq:Invalid time period input!\n");
    terminator ();
  }
}

/*
 * Connect to database with user/pass in C1
 */
void connectToDB (
  Config * C1)
{

  if (connectDB (C1) == NULL)
  {
    fprintf (stderr, "Connection to database failed!\n");
    terminator ();
  }
}

/*
 * Builds a SQL-query for low freqvent error search
 */
char *buildLFQuerySQL (
  char **in,
  char *fromdate)
{
  char *dbtable = NULL;
  char *sqlquery = NULL;
  /*
   * Set dbtable from inargument in[1]
   */
  if ((dbtable = setdbtable (in[1], in[5])) == NULL)
  {
    fprintf (stderr, "Could not set dbtable!\n");
  }

  if (existTable (dbtable))
  {
    /*
     * Create an sqlquery for data: [rindex, ltime, localtime, errorcode]
     */
    if ((sqlquery = lowfSqlQuery (dbtable, fromdate, in[2])) == NULL)
    {
      fprintf (stderr, "Could not create SQL-query for this table!\n");
    }
  }
  free (dbtable);
  return sqlquery;
}

/*
 * Collects all potential low freqvent errors in given timeperiod
 */
errorList *getErrors (
  char **in,
  char *fromdate,
  errorList * eListhead,
  int numberr,
  Config * C1)
{

  char *sqlquery = NULL;
  char time[9];
  int terlimit = 0;

  /*
   * List containing result of mysql query
   */
  MYSQL_POST *logList = NULL;

  for (;;)
  {

    sqlquery = buildLFQuerySQL (in, fromdate);

    /*
```

```
     * Mysql get complete interval for valid table
     */
    logList = getPosts (sqlquery);

    /*
     * Set upper limit to count timestamps for
     */
    terlimit = setLFErrorLim (QURYFACT, nr_set, numberr);

    /*
     * Store uniq errors in a list and count them as we go
     */
    eListhead = errUniq (logList, eListhead, terlimit);

    /*
     * Make logList empty again
     */
    logList = rm_SQL_List (logList);

    /*
     * Check if time to stop getting more data from database
     */
    strftime (time, 9, "%Y-%m", fr_);
    if (mktime (fr_) >= mktime (to_))
      break;
    if ((strncmp (time, in[2], 7)) == 0)
      break;

    /*
     * Move forward to next month
     */
    incr_month (1);
    *in[1] = set_tablemon (in[1]);

    /*
     * Connect to database again
     */
    connectToDB (C1);
  }

  free (sqlquery);
  return eListhead;
}

/*
 * Count the number of members in an errorList
 */
int countErrors (
  errorList * eListhead)
```

```
{
  int errorcounter = 0;
  errorList *eListcur = NULL;
  eListcur = eListhead;

  while (eListcur->ecode != NULL)
  {
    errorcounter += eListcur->hits;
    if (eListcur->next == NULL)
      break;

    eListcur = eListcur->next;
  }
  return errorcounter;
}

/*
 * Set upper limit to display for total set
 */
int setLFErrorLim (
  float FACTOR,
  int errorcounter,
  int numberr)
{
  float lim = 0;

  lim = (FACTOR * errorcounter);
  if (lim < numberr)
    lim = numberr;

  return (int) lim;
}

/*
 * Print a low freqvency report
 */
void printLFReport (
  errorList * eListcur,
  unsigned erlimit,
  unsigned short maxgrad,
  unsigned errorcounter,
  char *from,
  char **inarg)
{

  char reprow[50];
  unsigned int lowfqcounter = 0;
  short int idx;
  printf ("+-------------------------------------------+\n");
  printf ("|         LOW FREQUENT ERROR REPORT          |\n");
  printf ("+---------------------+---------------------+\n");
  printf ("|  From %s    |   To %s      | \n", from, inarg[2]);
  printf ("+---------------------+---------------------+\n\n");

  while (eListcur->ecode != NULL)
  {
    if (isValidPost (eListcur, erlimit, maxgrad))
    {

      printf ("+-------------------------------------------+\n");
      printf ("|            LOW FREQUENT ERROR              |\n");
      printf ("+-------------------------------------------+\n");
      snprintf
(reprow, 45, "| Error    : %s%s", eListcur->ecode, ESTRING);
      printf ("%s |\n", reprow);
      snprintf
(reprow, 45, "| Router nr: %d%s", *eListcur->rindex, ESTRING);
      printf ("%s |\n", reprow);
      snprintf
(reprow, 45, "| Router   : %s%s", eListcur->rname, ESTRING);
      printf ("%s |\n", reprow);
      snprintf
(reprow, 45, "| Num hits : %d%s", eListcur->hits, ESTRING);
      printf ("%s |\n", reprow);
      printf ("+-----------------+-------------------------+\n");
      printf ("|    Foundlist    |        Timestamps       |\n");
      printf ("+-----------------+-------------------------+\n");
      idx = 1;

      while (eListcur->timeList != NULL)
      {
        snprintf (reprow, 45, "| Num %d           | %s %s", idx,
                  ctime (&eListcur->timeList->time), ESTRING);
        printf ("%s |\n", reprow);
        idx++;
        eListcur->timeList = eListcur->timeList->next;
      }

      printf ("+-----------------+-------------------------+\n\n");
      lowfqcounter++;
    }
    if (eListcur->next == NULL)
      break;
    eListcur = eListcur->next;
  }

  printf ("Total number of error in set         = %d\n", errorcounter);
  printf ("Number of low frequency errors found = %d\n", lowfqcounter);
  printf ("Upper limit for hits was set to      = %d\n", erlimit);
  printf ("Max grade of severity was set to     = %d\n\n", maxgrad);
  printf ("Lowfreq LFRGFACT is set to           = %f\n", LFRQFACT);
  printf ("Presens of total time PRESFACT       = %f\n", PRESFACT);
  printf ("Factor of max irregularity IREGFACT  = %f\n\n", IREGFACT);
}

/*
 * Detects if a post is valid to be reported as a low freq error
 */
int isValidPost (
  errorList * post,
  unsigned int erlimit,
  unsigned short maxgrad)
{
  int isvalid = 0;
  int diff = 0;
  int dsum = 0;
  int idel = 0;
  time_t timelen;
  timeList *timecur;

  timecur = post->timeList;     /* First time in list */
  timelen = post->ltime - post->ftime;  /* Timesp. last found - first found */
  idel = (int) timelen / (post->hits - 1);      /* Calutate ideal regularity */

  if ((post->hits <= erlimit) && (post->hits > 1))
    if (post->grade <= maxgrad)
      if (interval * PRESFACT < timelen)
      {
        while (timecur->next != NULL)
        {
          diff = (timecur->next->time) - (timecur->time);
          dsum += abs (diff - idel);
          timecur = timecur->next;
        }
        if (dsum < timelen * IREGFACT)
          isvalid = 1;
      }
  return isvalid;
}

/*
 * Save errors uniq with statistic data from SQL-query
 */
/*
 * List from MYSQL-query must be in accending order of time
 */
```

```
errorList *errUniq (
  MYSQL_POST * logList,
  errorList * head,
  unsigned int lim)
{
  char *tmp;
  int inList = 0;
  MYSQL_POST *MYcur;
  errorList *cur, *newpost;
  timeList *timehead = NULL;
  timeList *timetemp = NULL;
  timeList *newtime = NULL;

  MYcur = logList;
  cur = head;

  while (MYcur != NULL)
  {
    while (cur->next != NULL || cur->ecode != NULL)
    {
      if ((strcmp (MYcur->ecode, cur->ecode) == 0) &&
          (*MYcur->rindex == *cur->rindex))
      {
        inList = 1;
        cur->hits += 1;
        cur->dateLast = MYcur->date;
        cur->timeLast = MYcur->time;
        cur->ltime = getUTC (MYcur);

        if (cur->hits <= lim)
        {                       /* Dont add to many already */
          /*
           * Add a timestamp for this hit
           */
          timetemp = cur->timeList;
          while (cur->timeList->next != NULL)   /* Move to end of timeList */
            cur->timeList = cur->timeList->next;

          newtime = (timeList *) malloc (sizeof (timeList));
          newtime->next = NULL;
          newtime->time = getUTC (MYcur);
          cur->timeList->next = newtime;
          cur->timeList = timetemp;
        }
      }
      if (inList)
        break;                  /* Just a bit more effective algorithm */
      if (cur->next == NULL)
        break;
    cur = cur->next;
  }
  if (!inList)
  {
    while (cur->next != NULL) /* Move to end of list */
      cur = cur->next;

    /*
     * Allocate memory for errorList with uniq errors
     */
    newpost = (errorList *) malloc (sizeof (errorList));
    newpost->next = NULL;
    newpost->ecode = MYcur->ecode;
    newpost->rindex = MYcur->rindex;
    newpost->rname = MYcur->rname;
    newpost->hits = 1;
    newpost->ftime = newpost->ltime = getUTC (MYcur);

    /*
     * Time settings for algorithm
     */
    timehead = (timeList *) malloc (sizeof (timeList));
    timehead->next = NULL;
    timehead->time = getUTC (MYcur);
    newpost->timeList = timehead;

    /*
     * Set grade of error suverity
     */
    tmp = strdup (MYcur->ecode);

    if (strtok (tmp, "-") != NULL)
      newpost->grade = (unsigned short) atoi (strtok (NULL, "-"));
    else
      newpost->grade = 7;     /* If no errorcode, set to WHAT? */

    free (tmp);

    /*
     * This values are only "for your information"
     */
    newpost->dateFirst = MYcur->date;
    newpost->timeFirst = MYcur->time;
    newpost->dateLast = MYcur->date;
    newpost->timeLast = MYcur->time;

    /*
     * Set time values
     */
```

```
      newpost->ftime = newpost->ltime = getUTC (MYcur);

      /*
       * Check if this is the first post
       */
      if (head->ecode == NULL)
        head = newpost;
      else
        cur->next = newpost;
    }
    inList = 0;                 /* Reset indikator */
    cur = head;                 /* Start all over with list of uniqlist  */
    MYcur = MYcur->next_post;   /* Move to the next post from database   */
  }
  return head;
}

/*
 * Remove the list(eListhead) and returns the head of an empty list
 */
errorList *rmeList (
  errorList * head)
{
  errorList *cur;
  timeList *tcur, *thead;

  while (head->next != NULL)
  {
    thead = head->timeList;
    if (thead != NULL)
      while (thead->next != NULL)
      {
        tcur = thead->next;
        thead->next = tcur->next;
        free (tcur);
      }
    free (thead);
    cur = head->next;
    head->next = cur->next;
    free (cur);
  }
  free (head);

  return (head = NULL);
}

/*
 * Saves fetched row into the list
 */
```

```
/*
 * In somewhat nerdy way but it works
 */
MYSQL_POST *setPostValue (
  MYSQL_POST * cur,
  MYSQL_ROW row)
{

  if (sizeof (row) != 4)
  {
    printf ("Invalid sql response\n");
    exit (-1);
  }

  if (cur == NULL)
  {
    printf ("List not created or defect (null)\n");
    exit (-1);
  }

  cur->rindex = (int *) malloc (sizeof (int));
  *cur->rindex = atoi (row[0]);
  cur->rname = (char *) strdup (row[1]);
  cur->time = (char *) strdup (row[2]);
  cur->date = (char *) strdup (row[3]);
  cur->ecode = (char *) strdup (row[4]);

  return cur;
}

/*
 * Get all data from MySQL
 * * Pre: sqlquery is a complete sql query
 * * Post: Local (MYSQL_POST*) head is set to the retrived  mysql data
 */
MYSQL_POST *getPosts (
  char *sqlquery)
{
  MYSQL_POST *current = NULL;
  MYSQL_RES *result;
  MYSQL_ROW row;
  unsigned int num_fields = 0;
  unsigned int num_rows = 0;

  head = mk_MYSQL_post (NULL);  /* Create the list */
  current = head;

  if (mysql_query (&connection, sqlquery))
    MYSQL_exiterr (1);
  if (!(result = mysql_use_result (&connection)))
    MYSQL_exiterr (2);

  num_fields = mysql_num_fields (result);

  if (num_fields > 0)
  {                             /* Like this, no extra post at end of list */
    row = mysql_fetch_row (result);
    current = setPostValue (current, row);

    while ((row = mysql_fetch_row (result)))
    {
      current = mk_MYSQL_post (current);
      current = setPostValue (current, row);
    }
  }
  else
  {
    printf ("No hits on your query, please try another time period \n");
    printf ("The version of MySQL used by this program is 3 and not 4, \n \
   in version 4 the UNION function is implemented and no extra \
   manual work has been done on this by us\n");
  }

  /*
   * mysql_num_rows() will not return the correct value until all
   * * the rows in the result set have been retrieved.
   * * This is why it's placed here
   */
  nr_set = num_rows = mysql_num_rows (result);

  if (mysql_errno (&connection))// mysql_fetch_row() failed due to an error
    MYSQL_exiterr (3);

  mysql_free_result (result);
  mysql_close (&connection);

  return head;
}
```

C.2 lowfreq.h

```
/***************************************************************************
 *                        lowferq.h  -  Headerfile                       *
 *                          -------------------                           *
 *  begin                : Thu Apr 09 2003                                *
 *  copyright            : (C) 2003 by Henrik Skantz & Mikael Böhm        *
 *                           D-Level Reserch                                *
 *  email               : hesk@linux.se, micko@linuxmail.org                *
 ***************************************************************************/
/***************************************************************************
 *   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 of the License, or     *
 *   (at your option) any later version.                                   *
 ***************************************************************************/
#ifndef LOWFREQ_FILE
#define LOWFREQ_FILE

/***************************************************************************/
/*
 * Non local variables
 */
/***************************************************************************/
struct tm *to_;         /* Time struct, keeps track of "to" date and time   */
struct tm *fr_;         /* Time struct, keeps track of "from" date and time */
struct tm *st_;         /* Time struct for "starttime", keep this static    */
unsigned int nr_set;    /* Number of rows on each query on database         */
time_t interval;        /* Total time interval of search                    */

/***************************************************************************/
/*
 * Constants
 */
/***************************************************************************/

/*
 * Algorithm filtering specific constants ********************************
 */
/*
 * Högre värde ger högre antal som övre gräns för lågfrekvent
 */
#define LFRQFACT     0.000025   /* Sets upper limit of whats lowfrekvency    */

/*
 * För varje tabell begränsande värde, ska matcha LFRQFACT
 */
#define QURYFACT     0.0003     /* Sets upper limit for table lowfreq.limit  */

/*
 * Hur mycket av den totala tidsperioden som måste vara representerad
 */
/*
 * Max 1, ger 0 träffar, Min 0, ger inga begränsningar
 */
#define PRESFACT     0.15       /* Part of total period to be represented    */
```

```
/*
 * Anger hur felens spridning ska sållas
 */
/*
 * 0 ger inga träffa, 1 medför inga begränsningar
 */
#define IREGFACT     0.66       /* Factor for maximum irregularity of errors */
/***************************************************************************/

#define USAGELOW "Usage: lowfreq <start-day [2003-03-21]> \
<end-day [2003-03-31]> <min reported errors [1 - inf]> \
<max grade to report[0 -7] <system[cisco]>\n"

#define RMARG 15                /* Right margin prints */
#define ESTRING "                                "      /* For prints */

/***************************************************************************/
/*
 * Type definitions
 */
/***************************************************************************/
struct errorList
{
  char *ecode;                  /* Error              */
  int *rindex;                  /* Router             */
  char *rname;                  /* Router name        */
  unsigned int hits;            /* Times found        */
  unsigned short grade;         /* Suverity           */
  char *dateFirst;              /* Date first found   *//* Youst Info */
  char *timeFirst;              /* Time first found   *//* Youst Info */
  char *dateLast;               /* Date last found    *//* Youst Info */
  char *timeLast;               /* Time last found    *//* Youst Info */
  time_t ftime;                 /* First time found   */
  time_t ltime;                 /* Last time found    */
  timeList *timeList;           /* List of timestamps */
  errorList *next;              /* Next post in list  */
};

struct timeList
{
  time_t time;                  /* Time for error/rt  */
  timeList *next;               /* Next pos in list   */
};

/***************************************************************************/
/*
 * Function prototypes
 */
```

```
/****************************************************************************/
/*
 * Process flow ctrl function
 */
void lowfreq (
  char **in);

/*
 * Allocates new memory for array at pos and return as char
 */
char *getArrayPos (
  char **in,
  int pos);

/*
 * Setting the global time objekts values
 */
void setTimePeriod (
  char *fr,
  char *to);

/*
 * Connect to database with user/pass in C1
 */
void connectToDB (
  Config * C1);

/*
 * Collects all potential low freqvent errors in given timeperiod
 */
errorList *getErrors (
  char **in,
  char *fromdate,
  errorList * eListhead,
  int numberr,
  Config * C1);

/*
 * Count the number of members in an errorList
 */
int countErrors (
  errorList * eListhead);

/*
 * Set upper limit to display for total set
 */
int setLFErrorLim (
  float FACTOR,
  int errorcounter,
  int numberr);

char *buildLFQuerySQL (
  char **in,
  char *fromdate);

/*
 * Detects if a post is valid to be reported as a low freq error
 */
int isValidPost (
  errorList * post,
  unsigned int erlimit,
  unsigned short maxgrad);

/*
 * Print a low freqvency report
 */
void printLFReport (
  errorList * eListcur,
  unsigned erlimit,
  unsigned short maxgrad,
  unsigned errorcounter,
  char *from,
  char **inargs);

/*
 * Save errors uniq with statistic data from SQL-query
 */
errorList *errUniq (
  MYSQL_POST * logL,
  errorList * h,
  unsigned int lim);

/*
 * Remover of errorList
 */
errorList *rmeList (
  errorList * head);

/*
 * Fetch posts fr. query
 */
MYSQL_POST *getPosts (
  char *sqlquery);

/*
 * Set values
 */
MYSQL_POST *setPostValue (
  MYSQL_POST * cur,
  MYSQL_ROW row);

/*
 * Terminate the process
 */
void terminator (
  );

#endif
```

Appendix D

Common files

D.1 timeCommon.c

D.2 timeCommon.h

D.3 readconf.c

```
/***************************************************************************
                          mylogconf.c  -  description
                             -------------------
    begin                : Sat Nov 9 2002
    copyright            : (C) 2002 by Mikael Bï£¡m & Henrik Skantz
    email                : micko@linuxmail.org, hesk@linux.se
 ***************************************************************************/

/***************************************************************************
 *                                                                         *
 *   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 of the License, or     *
 *   (at your option) any later version.                                   *
 *                                                                         *
 ***************************************************************************/
#include "header.h"

/*
 * Retrives system parameters from given configuration file.
 * String cfile. This will be put in the struct lc.
 * Returns:
 * Read parameters are put in a record lc which is returned.
 * Opon failures with file, errormessage is displayed and the
 * program is terminated
 */
```

```
Config *readConf (
  String cfile,
  Config * lc)
{

  int fd;                       /* File descriptor         */
  char row[CONF_ROW];           /* Read row from file      */
  struct stat file_info;        /* File info               */
  unsigned int offset = 0;      /* File offset             */
  unsigned int vconfl = 0;      /* Counter for vaild lines */
  String str;                   /* Pointer to parameter    */

  fd = OpenFile (cfile, O_RDONLY);

  /*
   * Get information about the file.(fsize)
   */
  if (fstat (fd, &file_info) == FAIL)
  {
    perror ("fstat");
    CloseFile (fd, cfile);
    terminator ();
  }

  while (file_info.st_size > (int) offset)
  {
    GetRow (fd, row, &offset);
    if ((str = strstr (row, "mysqlhost=")) != NULL)
      strcpy (lc->dbhost, str);
    else if ((str = strstr (row, "db=")) != NULL)
      strcpy (lc->db, str);
    else if ((str = strstr (row, "dbuser=")) != NULL)
      strcpy (lc->dbuser, str);
    else if ((str = strstr (row, "dbpass=")) != NULL)
      strcpy (lc->dbpass, str);
    else if ((str = strstr (row, "pipefile=")) != NULL)
      strcpy (lc->pipe, str);
    else
      continue;

    vconfl += 1;
  }
  CloseFile (fd, cfile);

  if (vconfl != NCONF_OPTIONS)
  {
    perror ("No valid config file");
    terminator ();
  }
  /*
   * Remove infront text
   */
  strcpy (lc->dbhost, strtok (strstr (lc->dbhost, "="), "="));
  strcpy (lc->db, strtok (strstr (lc->db, "="), "="));
  strcpy (lc->dbuser, strtok (strstr (lc->dbuser, "="), "="));
  strcpy (lc->dbpass, strtok (strstr (lc->dbpass, "="), "="));
  strcpy (lc->pipe, strtok (strstr (lc->pipe, "="), "="));

  return lc;
}
```

D.4 readconf.h

```
/***************************************************************************
 *                      readconf.h  -  Headerfile                        *
 *                         -------------------                             *
 *  begin              : Thu Sep 05 2002                                   *
 *  copyright          : (C) 2002 by Henrik Skantz & Mikael BÃũhm          *
 *                           Bachelors project - KaU fall 2002             *
 *  email              : hesk@linux.se, micko@linuxmail.org                *
 ***************************************************************************/
/***************************************************************************
 *   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 of the License, or     *
 *   (at your option) any later version.                                   *
 ***************************************************************************/
#ifndef READCONF_FILE
#define READCONF_FILE

/***************************************************************************/
/*
 * Constants
 */
/***************************************************************************/
#define NCONF_OPTIONS   5
#define CONF_ROW      300
#define DBNAME_LEN     30
#define TABLENAME_LEN  30
#define HOSTNAME_LEN   50
#define PATH_LEN       50
#define USER_LEN       30
#define PASS_LEN       30
#define SYSTEM_LEN     20

/***************************************************************************/
/*
```

```
 * Type definitions
 */
/***************************************************************************/
struct Config
{
  char dbhost[HOSTNAME_LEN];
  char db[DBNAME_LEN];
  char dbtable[TABLENAME_LEN];
  char dbuser[USER_LEN];
  char dbpass[PASS_LEN];
  char system[SYSTEM_LEN];
  char pipe[PATH_LEN];
};

/***************************************************************************/
/*
 * Function prototypes
 */
/***************************************************************************/
Config *readConf (
  String cfile,
  Config * lc);

#endif
```

D.5 listCommon.c

```
/***************************************************************************
                          listCommon.c  -  description
        ----<<0>>--- * List functions for diffusion *  ---<<0>>---
                             -------------------
    begin                : Wen Apr 10 12:00:00 CET 2003
    copyright            : (C) 2003 by Mikael Böhm & Henrik Skantz
    email                : micko@linuxmail.org, hesk@linux.se
 ***************************************************************************/
/***************************************************************************
 *   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 of the License, or     *
 *   (at your option) any later version.                                   *
 ***************************************************************************/

#include "header.h"

/*
 * Make a new tail for the list
 */
/*
```

```
 * Use NULL as argument when creating a new list.
 */
MYSQL_POST *mk_MYSQL_post (
  MYSQL_POST * current_post)
{
  MYSQL_POST *new_post;

  if ((new_post = (MYSQL_POST *) malloc (sizeof (MYSQL_POST))) == NULL)
    fprintf (stderr, "Allocation of memory for new MYSQL_POST failed!\n");

  if (current_post != NULL)
    current_post->next_post = new_post;

  new_post->next_post = NULL;   /* the new node will be the tail */

  return new_post;
}

/*
 * Make a new tail for the list
 */
/*
 * Use NULL as argument when creating a new list.
 */
winError *mk_winError_post (
  winError * current_post)
{
  winError *new_post;

  if ((new_post = (winError *) malloc (sizeof (winError))) == NULL)
    fprintf (stderr, "Allocation of memory for new MYSQL_POST failed!\n");

  if (current_post != NULL)
    current_post->next_post = new_post;

  new_post->next_post = NULL;   /* the new node will be the tail */

  return new_post;
}

/*
 * Returns a allocated diffList post or NULL
 */
diffList *mk_diffList_post (
  )
{
  diffList *new_post;
  return (new_post = (diffList *) malloc (sizeof (diffList)));
}
```

```
/*
 * Returns a allocated rindexList post or NULL
 */
rindexList *mk_rindexList_post (
  )
{
  rindexList *new_post;
  return (new_post = (rindexList *) malloc (sizeof (rindexList)));
}

/*
 * Remove the entire SQL-result table, head is in non-local-env
 */
void remSQLtable (
  )
{
  MYSQL_POST *cur;
  while (head != NULL)
  {
    cur = head->next_post;
    free (head);
    head = cur;
  }
}

/*
 * Remove the list (MYSQL_POST*) and returns the head of an empty list
 */
/*
 * This is a variant for non global lists
 */
MYSQL_POST *rm_SQL_List (
  MYSQL_POST * head)
{
  MYSQL_POST *cur;

  while (head->next_post != NULL)
  {
    cur = head->next_post;
    head->next_post = cur->next_post;
    free (cur);
  }
  head = NULL;
  return head;
}

/*
 * Remove the list (winError*) and returns the head of an empty list
 */
winError *removeList (
  winError * head)
{
  winError *cur;
  while (head->next_post != NULL)
  {
    cur = head->next_post;
    head->next_post = cur->next_post;
    free (cur);
  }

  return head;
}

/*
 * Prints one (MYSQL_POST*) post ref. by cur
 */
void print_post (
  MYSQL_POST * cur)
{
  if (cur != NULL)
  {
    printf ("-------------------------------------------------------------\n|");
    printf (" %d |", *cur->rindex);
    fflush (stdout);
    printf (" %s |", cur->time);
    fflush (stdout);
    printf (" %s |", cur->date);
    fflush (stdout);
    printf (" %s \n", cur->ecode);
  }
}

/*
 * Returns next (MYSQL_POST*) post in the list ref. by 'current_post'
 */
MYSQL_POST *next_post (
  MYSQL_POST * current_post)
{
  return current_post->next_post;
}
```

D.6 listCommon.h

```
/***************************************************************************
 *   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 of the License, or     *
 *   (at your option) any later version.                                   *
 ***************************************************************************/

#ifndef LISTC_FILE
#define LISTC_FILE
/***************************************************************************/
/*
 * Non local variables
 */
/***************************************************************************/
MYSQL_POST *head;          /* Head of SQL table query */
winError *winEhead;        /* Head of window list */
MYSQL_ERROR *err_head;     /* Head of a potential error spread */
MYSQL_POST *toffset;       /* Table offset , where to start in next search */
/***************************************************************************/
/*
 * Type definitions
 */
/***************************************************************************/
struct diffList
{
  char *ecode;
  char *time;
  char *date;
  rindexList *rindexList;
  diffList *next;
};
struct rindexList
{
  int *rindex;
  rindexList *next;
};
struct MYSQL_ERROR
{
  char *error;
  MYSQL_ERROR *next_post;
};
struct winError
{
  MYSQL_POST *mysqlPost;
  winError *next_post;
};
struct MYSQL_POST
{
  int *rindex;
  char *time;
  char *date;
  char *ecode;
  char *rname;
  MYSQL_POST *next_post;
};

/****************************************************************************/
/*
 * Function prototypes
 */
/****************************************************************************/

MYSQL_POST *mk_MYSQL_post (
  MYSQL_POST * current_post);
MYSQL_POST *next_post (
  MYSQL_POST * current_post);
MYSQL_POST *rm_SQL_List (
  MYSQL_POST * head);

winError *removeList (
  winError * head);
winError *mk_winError_post (
  winError * current_post);
diffList *mk_diffList_post (
  );
rindexList *mk_rindexList_post (
  );

void print_post (
  MYSQL_POST *);
void remSQLtable (
  );

#endif
```

D.7 filectrl.c

```
/***************************************************************************
                          filectrl.c  -  description
                             -------------------
    begin                : Thu Sep 12 2002
    copyright            : (C) 2002 by Mikael Bï£¡m & Henrik Skantz
    email                : micko@linuxmail.org, hesk@linux.se
 ***************************************************************************/
/***************************************************************************
 *   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 of the License, or     *
 *   (at your option) any later version.                                   *
 ***************************************************************************/
```

```
#include "header.h"

/*
 * Reads from fd until \n or EOF.
 * Returns:
 * Read data from fd is returned and the offset
 * is advanced with the number of bytes read.
 * On read error row is set to be empty.
 */
String GetRow (
  int fd,
  char *row,
  unsigned int *offset)
{

  int nread;
  char c[1];

  bzero (row, strlen (row));

  if (lseek (fd, *offset, SEEK_SET) == FAIL)
  {
    fprintf (stderr, "line %3d : lseek error\n", __LINE__);
    exit (FAIL);
  }

  while ((nread = read (fd, c, 1) > 0) && (*c != '\n'))
  {
    strncat (row, c, 1);
    *offset += 1;
  }
  /*
   * Advance to pass \n for next ev. read
   */
  *offset += 1;

  if (nread == FAIL)
  {
    fprintf (stderr, "line %3d : read error\n", __LINE__);
    bzero (row, strlen (row));
  }
  return row;
}

/*
 * Closes the file fname on filedesc. fd.
 * Filename should be provided for better information
 * to the user.
 */
```

```
void CloseFile (
  int fd,
  String fname)
{
  if (close (fd) == FAIL)
    fprintf (stderr, "line %3d : Could not close %s\n", __LINE__, fname);
}

/*
 * Opens file "fname" in mode "mode".
 * Returns: On success a filedescriptor fd is returned,
 * else errormessage and termination
 */
int OpenFile (
  String fname,
  int mode)
{
  int fd;
  if ((fd = open (fname, mode)) == FAIL)
  {
    fprintf
      (stderr, "line %3d : No valid fileref from inputfile : %s\n",
       __LINE__, fname);
    terminator ();
  }
  return fd;
}
```

D.8 filectrl.h

```
/***************************************************************************
                          filectrl.h  -  description
                             -------------------
    begin                : Thu Sep 12 2002
    copyright            : (C) 2002 by Mikael Böhm & Henrik Skantz
    email                : micko@linuxmail.org, hesk@linux.se
 ***************************************************************************/
/***************************************************************************
 *   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 of the License, or     *
 *   (at your option) any later version.                                   *
 ***************************************************************************/
#ifndef FILE_HEADER
#define FILE_HEADER

/****************************************************************************/
/*
```

```
 * Function prototypes
 */
/****************************************************************************/
String GetRow (
  int fd,
  char *row,
  unsigned int *offset);
void CloseFile (
  int fd,
  char *fname);
int OpenFile (
  String fname,
  int mode);

#endif
```

D.9 mysqlQ.c

```
/***************************************************************************
                         mysqlQ.c  -  MySQL backend
                             -------------------
    begin                : Thu Sep 12 2002
    copyright            : (C) 2002 by Mikael Bï£¡m & Henrik Skantz
    email                : micko@linuxmail.org, hesk@linux.se
 ***************************************************************************/
/***************************************************************************
 *   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 of the License, or     *
 *   (at your option) any later version.                                   *
 ***************************************************************************/
#include "header.h"

/*
 * Standard sql-query for low freqency analycis
 * Returns : sql query string
 */
String lowfSqlQuery (
  char *dbtable,
  char *lowdate,
  char *highdate)
{
  char *querystr;

  /*
   * Allocate memory for SQL-query
   */
  if ((querystr = (char *) malloc (350)) == NULL)
  {
    fprintf (stderr, "Allocation of memory failed!\n");
    terminator ();
  }

  sprintf
(querystr, "SELECT rindex,routers.name as rname,ltime,localdate,errorcode \
  FROM `%s` inner join routers on `%s`.rindex = routers.routernr \
  where localdate >= \"%s\" and localdate <= \"%s\" \
  order by localdate, ltime", dbtable, dbtable, lowdate,
           highdate);

  return querystr;
}

/*
 * Creates a SQL query for diffusion.
 * Returns: The SQL query string.
 */
String diffSqlQuery (
  char *dbtable)
{
  char *querystr;
  char time[12], time2[12];

  /*
   * Allcate memory for SQL-query
   */
  if ((querystr = (char *) malloc (350)) == NULL)
  {
    fprintf (stderr, "Allocation of memory failed!\n");
    terminator ();
  }

  strftime (time, 12, "%Y-%m-%d", fr_);
  strftime (time2, 12, "%Y-%m-%d", to_);

  sprintf (querystr, "SELECT rindex,ltime,localdate,errorcode \
  FROM `%s` where localdate >= \"%s\" and localdate <= \"%s\" \
  order by localdate, ltime", dbtable, time, time2);

  return querystr;
}

/*
 * Creates a SQL query for rekerr.
 * Returns: The SQL query string.
 */
String rekSqlQuery (
  char *dbtable)
{
  char *querystr;
  char time[12], time2[12];
  int diff = -1;

  /*
   * Allcate memory for SQL-query
   */
  if ((querystr = (char *) malloc (350)) == NULL)
  {
    fprintf (stderr, "Allocation of memory failed!\n");
    terminator ();
  }
  diff = mktime (to_) - mktime (fr_);

  strftime (time, 12, "%Y-%m-%d", fr_);
  strftime (time2, 12, "%Y-%m-%d", to_);

  if (diff != 0)
  {
    sprintf
(querystr, "SELECT rindex,routers.name as rname,ltime,localdate,errorcode \
    FROM `%s` inner join routers on `%s`.rindex = routers.routernr  where \
    localdate >= \"%s\" and localdate <= \"%s\" order by localdate, ltime",
              dbtable, dbtable, time, time2);
  }
  else
  {
    sprintf
(querystr, "SELECT rindex,routers.name as rname,ltime,localdate,errorcode \
          FROM `%s` inner join routers on  `%s`.rindex = routers.routernr \
  where localdate = \"%s\" order by localdate, ltime", dbtable,
              dbtable, time);
  }

  return querystr;
}

/*
 * Sets up a safe SSL connection to a local or remote database.
 * The connection is establised from data read from config file.
 * Returns:
 * If connection to db is successful the connection returns else the
 * the program will terminate with errormessage
 */
MYSQL *connectDB (
  Config * conf)
{
  MYSQL *con;

  /*
   * Setup a the mysql connection with SSL
   */
  con = mysql_init (&connection);

  mysql_real_connect (&connection, conf->dbhost, conf->dbuser, conf->dbpass,
                      conf->db, 0, NULL, CLIENT_SSL);

  if (con == NULL)
    printf (mysql_error (&connection));

  return con;
}

/*
 * Close  connection to db
 */
void closeDB (
  )
{
  mysql_close (&connection);
}

/*
 * Runs a SQL query on selected database connection
 * A connection must exist to a db
 */
void InsToDb (
  String sqlquery,
  Config * conf)
{
  int state;
  int retries = 0;

  state = mysql_query (&connection, sqlquery);

  /*
   * Query failed (timedout?)=> retry connection then try again
   */
  while ((state != 0) && (retries < MAXRETRIES))
  {
    retries++;
    connectDB (conf);
    sleep (1);
    state = mysql_query (&connection, sqlquery);
  }
  /*
```

```
   * Connection has dropped or server outage
   */
  if (state != 0)
  {
    printf (mysql_error (&connection));
    terminator ();
  }
}

/*
 * Checks if  a table exsists in the db
 * A connection must exist to a db
 * Returns :
 * If the table match the simple regular expression in [table] the
 * function returns true, else false
 */
boolean existTable (
  String table)
{
  MYSQL_RES *result;
  boolean boolval;

  if ((result = mysql_list_fields (&connection, table, NULL)) == NULL)
    boolval = false;
  else
    boolval = true;

  /*
   * Must free memory according to MySQL API
   */
  mysql_free_result (result);
  return boolval;
}

/*
 * Defines which dbtable to use, returns it
 * memory is allocated for it !
 */
char *setdbtable (
  char *in_a,
  char *in_b)
{
  char *table;
  char *yymm = NULL;

  if ((table = (char *) malloc (30)) == NULL)
    fprintf (stderr, "Allocation of memory  failed!\n");

  yymm = (char *) malloc (sizeof (in_a) + 1);
  strcpy (yymm, in_a) /* Table date */ ;
  strcpy (table, in_b);         /* System, e.g. cisco */
  strcat (table, SEPARATOR);

  /*
   * Valid to 2099 HOWTO DO THIS SMART?
   */
  if ((yymm = strtok (yymm, "-")) == NULL)
  {
    free (table);
    printf ("Invalid date format\n");
    exit (-1);
  }
  else
  {
    yymm += 2;                  //yymm++;yymm++;
    strcat (table, yymm);
  }
  if ((yymm = strtok (NULL, "-")) == NULL)
  {
    free (table);
    printf ("Invalid date format\n");
    exit (-1);
  }
  else
    strcat (table, yymm);

  return table;
}

/*
 * A helping tool for mysql errors
 */
void MYSQL_exiterr (
  int exitcode)
{
  fprintf (stderr, "%s\n", mysql_error (&connection));
  exit (exitcode);
}

/*
 * Upon defined signals this function will
 * take closing and cleaning actions.
 */

void terminator (
  )
{
  time_t Epoch;
  struct tm *t;
  Epoch = time (NULL);
  t = localtime (&Epoch);

  printf ("Terminating %d:%d:%d\n", t->tm_hour, t->tm_min, t->tm_sec);
  closeDB ();
  exit (0);
}
```

D.10 mysqlQ.h

```
/***************************************************************************
 *                      mysqlQ.h  -  Headerfile                         *
 *                         -------------------                            *
 *  begin               : Thu Sep 05 2002                                 *
 *  copyright           : (C) 2002 by Henrik Skantz & Mikael BÃũhm          *
 *                           Bachelors project - KaU fall 2002            *
 *  email               : hesk@linux.se, micko@linuxmail.org              *
 ***************************************************************************/
/***************************************************************************
 *   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 of the License, or     *
 *   (at your option) any later version.                                   *
 ***************************************************************************/
#ifndef MYSQLQ_FILE
#define MYSQLQ_FILE

/***************************************************************************/
#define ROUTERTAB         "routers"  /* Must be predefined in mysql db     */
#define FIELDRNR          "routernr" /* Fieldname for index in routertable */
#define FIELDN            "name"     /* Fieldname for name in routertable  */

/***************************************************************************/
/*
 * Constants
 */
/***************************************************************************/
#define MAXRETRIES           3
#define QSIZE               1000

/***************************************************************************/
/*
 * Function prototypes
 */
/***************************************************************************/
String lowfSqlQuery (
  char *dbtable,
  char *lowdate,
  char *highdate);
String diffSqlQuery (
  char *dbtable);
String rekSqlQuery (
  char *dbtable);
MYSQL *connectDB (
  Config * conf);
void closeDB (
  );
void terminator (
  );
void InsToDb (
  char sqlquery[],
  Config * conf);
boolean existTable (
  String table);
char *setdbtable (
  char *in_a,
  char *in_b);
void MYSQL_exiterr (
  int exitcode);
#endif
```

D.11 makefile

```
############################################################################
#    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 of the License, or     #
#    (at your option) any later version.                                   #
############################################################################

# .PHONY will run the commands regardless of whether there is a file
# named 'clean' or 'all'.
.PHONY: all clean

### NO '/' AT END OF PATH ####################
# Binary install dest.
BINDEST= /usr/bin

# Extra includes and libs
MYSQLINC= /usr/include/mysql
MYSQLLIB= /usr/lib/mysql

# Compiler
CXX= gcc
```

```
CXXFLAGS= -O2 -g3 -W -Wall -Wundef -Wpointer-arith -Wsign-compare -Wmissing-declarations
# För ANSI C: Lägg till  -pedantic

# Link
LIBS= -lz -lmysqlclient

INCDIR= -I$(MYSQLINC)
LIBDIR= -L$(MYSQLLIB)

INCALL = mysqlQ.o readconf.o filectrl.o listCommon.o timeCommon.o

#objects = $(patsubst %.c,%.o,$(wildcard *.c))
objects1 = diffusion.o $(INCALL)
objects2 = lowfreq.o $(INCALL)
objects3 = rekerr.o rekerrMySQL.o $(INCALL)

all: diffusion lowfreq rekerr dist

%.o: %.c
$(CXX) -c $(CXXFLAGS) $(INCDIR) -o $@ $<

diffusion: $(objects1)
$(CXX) -o diffusion $(objects1) $(LIBDIR) $(LIBS)

lowfreq: $(objects2)
$(CXX) -o lowfreq $(objects2) $(LIBDIR) $(LIBS)

rekerr: $(objects3)
$(CXX) -o rekerr $(objects3) $(LIBDIR) $(LIBS)

install:
cp  diffusion rekerr lowfreq $(BINDEST)
uninstall:
rm -f $(BINDEST)/diffusion $(BINDEST)/rekerr $(BINDEST)/lowfreq
dist:
rm -f $(objects1) $(objects2) $(objects3) *~
clean:
rm -f $(objects1) $(objects2) $(objects3) *~ diffusion lowfreq rekerr
```

Appendix E

Script files

E.1 lowfreq.sh

```
#!/bin/sh

date1=‘date +%Y’-’%m’-’%d --date=’30 days ago’‘
date2=‘date +%Y’-’%m’-’%d‘

#echo $date1
#echo $date2

/usr/bin/lowfreq $date1 $date2 5 6 cisco
```