UNIVERSITY OF STUTTGART

DEVELOPMENT OF AN ADAPTIVE LEARNING NETWORK-FAILURE DETECTION SYSTEM

Final Project: Telecommunication Engineering

Iñigo Ucero Aristu

9/30/2008

Supervisors: Antonio Cuevas & Patrick Mandic

Abstract

The purpose of this work consists in finding a method able to detect anomalies and adapt to new behaviours in an IP-network. Here is possible to find a work of investigation about the different anomaly detection systems.

During the introduction and the first points, the audience could realize how the field of anomaly detection is divided in two big groups: Anomaly detection systems based in rules, and adaptive anomaly detection systems. These two ideas will be discuss, and some examples of each technology are given.

In the case of adaptive anomaly detection, there are some techniques proposed. One of them is developed during the rest of the work. I wanted to implement this technique in a small network property of the RUS department of the University of Stuttgart. The implementation, the problems found and additional information can be found in this report. Later the tests and results applied make us think if the behaviour is correct or not.

The discussion and the ideas obtained can be found at the end of the report, where is possible to find also the advices for future developers.

This report has been designed as a guide for developers which lend them to avoid too much time in understanding the present algorithm. The aid of this work is to be considered as a helper reference to people who wants to implement this kind of systems.

Contents

List of Figures

List of Tables

Glossary

Anomaly Detection *System which detects anomalies due to enemy's actions and networks failures*

Arbor Peak-Flow *Network monitoring tool property of Arbor Networks*

Arbor Peak-Flow *Software which manages the network traffic and is based in strict rules that can be modelled by the manager*

ATF *Arbor Peak-Flow file which includes information about rules to be installed*

Cisco Systems *Cisco is an Enterprise which develops software and hardware for networks.*

GLR *Generalized likelihood ratio*

GLRT *Generalized likelihood ratio test*

IDS *Intrusion Detection System*

Intrusion Detection *System which detects anomalies due to enemy's actions and is based in static rules and are specify for only one anomaly*

MIB *Management Information Base*

OS *Operation System*

SNMP *Simple Network Management Protocol*

SYSLOG *Standard for forwarding log messages in IP networks.*

1 Introduction

1.1 Rationale

Network security and robustness have become two of the bases in network design. Both aspects let the users to communicate with guaranties in the nets all around the world. Designers and intruders know this, so this part of the design of networks is always adapting to new circumstances.

Apart from the attack detection, which is not the main purpose of this report, there have been many designs throughout the years since the advent of the communications networks. Developers in this field have been working on them trying to find the final system which will solve all the problems in the present and in the future. This is a hard work which has not been closed yet. For the moment, the only thing that designers can do is to approximate solutions to this ambitious idea. Techniques coming from the signal processing field started to be used, leading to more powerful systems.

A network anomaly is a sudden and short-lived deviation from the normal operation of the network. Some anomalies are deliberately caused by intruders with malicious intent such as a denial-of-service attack in an IP network, while others may be purely an accident. Quick detection is needed to initiate a timely response.

The problem of non-adaptive algorithms is such that the algorithms have to have clear rules which let them to identify an anomaly. Rules must describe all the anomalous behaviour in the network for which managers must specify them very carefully. The hard work identifying the best rules is the task I want to avoid with my work. This is the field where I will develop all my experiments.

1.2 Objectives and task plan

The purpose of this work is to find an adaptive anomaly-detection system, specify its advantages and disadvantages, and finally implement it in a real environment.

The algorithm must be able to detect network anomalies specifying the area where it happened but not the kind of anomaly. This capability can be obtained with other algorithms, but I am not developing them in this report.

The task plan that I need to carry out is summarized below:

a) Study the problems in the networks and the best solutions for them.
b) Choose an interesting adaptive anomaly-detection algorithm.
c) Specify the advantages and disadvantages of this algorithm.
d) Implement it.
e) Present the results and discuss them.

The development is carried out in the University of Stuttgart, more specifically in the RUS department. All the machines and equipment are supply by RUS. I had to limit my experiments to the available equipment in the department.

1.3 Document outline

This work is divided in sections which describe the fields of action. The report presents different solutions to the problem of anomaly detection, and specifies one which has been chose in order to be implemented.

Section 1 narrates the problem which must be solved and a brief description of the course of action in order to solve the problem.

The solutions given along the years and the current situation are presented in Section 2. There is possible to find the past situation of this kind of systems and how they have been developed during the last years. As an example of the previous kind of algorithms used even these days, ArborPeakflow is presented. The solution given by this software is still useful in many situations. At the end of Section 2 the audience can find adaptive algorithms which try to improve the efficiency of the earlier algorithms like ArborPeakflow. Besides the algorithm developed in this work is exposed there.

Section 3 contains the full description of the algorithm implemented composed by the theoretical issue and the environment. The theoretical part explains the work carried out by Marina Thottan and Chuanyi Ji (1). The environment is the natural situation where the algorithm works and how it interacts with the network. At the end of the section the problems found during the implementation of the algorithm are shown.

The tests carried out can be found in Section 4. At the beginning the test environment is explained. It is not exactly the same as the environment proposed in Section 3, but it simulates the behaviour in small size in order to extrapolate the results to the natural environment. After that the tests proposed and the results are shown. This section finishes with an analysis of the results, comparison with other possible solutions and the confidence of the environment approach given for the tests.

The conclusion and further ideas are contained in Section 5. There is a discussion about the viability of the algorithm proposed and future work based on this current development.

2 Analysis of solutions for network-failure

In the last fifteen years, systems have changed the idea of detection of failures in the networks. Before 2000 the idea was *intrusion detection* but nowadays it is *anomaly detection* (2). They are different philosophies.

The *intrusion detection* acts with static rules and defined assessments, trying to solve problems previously identified. In the other hand, the *anomaly detection* acts when there is an abnormal behaviour of the network. In this case, normal network behaviour is known, and every different situation is identified as a failure. Thus the system doesn't need to know every single fault. As it is explained below, my work deals with this kind of systems.

The conception of an *anomaly detection system* is based most of the cases in unsupervised data mining. Data mining is a signal processing technique that picks up relevant information from considerable amount of data. Moreover it is unsupervised because the algorithm doesn't know which will be the result of its application. Data mining is very useful in many fields of sciences. It is important in business intelligence organizations and financial analysis too.

2.1 Previous solutions. Arbor Peakflow

I am focusing now in the detection systems before the advent of adaptive detection algorithms. These systems work in many networks even these days. They have improved along the time, but since their conception they have followed the same guidelines.

They consist in some fixed thresholds, which when exceeded, alarms are showed. This is good in many cases, but not always. The problem lies in new attacks. When a network is designed is basic to know which are the common fails and attacks. After that, thresholds can be chosen. But these thresholds do not work well with every new problem in the network.

There is a kind of algorithms based in rules which try to detect intrusions in the system. These systems are called IDS (Intrusion Detection System). Here I show a brief introduction to IDS

- An IDS or Intrusion Detection System is a security tool that tries to detect or monitor events in a particular computer system or computer network which attempt to compromise the security of that system.
- The IDS seeks pre-defined patterns involving any kind of suspicious or malicious activity on our network or host.
- The IDS brings security to our capacity for prevention and early warning against any suspicious activity. They are not designed to stop an attack, although they can generate certain types of response to them.
- IDS: increases the security of our system, monitors traffic on our network, examines packages analyzing data in search of suspects and detect the early stages of any attack such as the analysis of our network, scanning ports, etc.

We are not focusing on them, but we can realize that they are still alive and solve too many problems.

The aim of the next point is to give an idea of this kind of systems, offering a brief description of the software Arbor Peakflow.

2.1.1 Arbor Peakflow.

RUS department manages the network with the help of Arbor Peakflow. This is the main application for intrusion and anomaly detection.

This tool is powerful software basing its conception on the idea: *intrusion detection*. The management of this tool is not so complicated because it has an intuitive interface. But the problem is that rules must be updated often in order to obtain a good behaviour of the software. These rules are, in simple words, thresholds that detect all kind of failures in the network.

*Arbor Peak-Flo*w is able to detect attacks of phishing, pharming, worms and spyware, as well as network failures. For new attacks, network managers must be sure that an attack behaves in a special way. If programmers do not deliver so much about it, the rules can be wrong and damage the normal behavior of the network.

Some aspects of *Arbor Peak-Flow* are:

a) **Behaviour**: Several events that can be detected.
b) **Rules**: When a strange behaviour is detected, rules specify the actions that must be taken.
c) **Alerts**: Notifications are sent when an alarm is activated.
d) **Notification**: There are three ways of notification, e-mail, SNMP, and SYSLOG traps.
e) **Policy**: All the behaviours, rules and problem solutions that *Arbor Peak-Flow* generates for the network which is monitored.

When *Arbor Peak-Flow* is installed for the first time, it creates common rules for the network. These rules can be modified later on, or corrected. It avoids too much work for the managers of the network. It lets adding of preconfigured rules from ATF files also.

This tool let the managers check the traffic in the network, and see where the users go. It is a complex tool with a huge sort of applications.

One of these useful tools is the presentation of network statistics in real time. The statistics can be defined in a range of time and the system presents them as graphics of traffic, behaviors… This information can be read in a table also, and it can be sort in different ways as destination IP address of the flow, or number of accesses to a web page.

This has been a small description of the capacity handled by *Arbor Peak-Flow*. But for more information audience can search in the references (3) (4) or in the *Arbor* web page (5).

2.2 Adaptive algorithms

The next generation of algorithms is composed by those algorithms able to memorize the last information received and to process it, in order to get an adaptive technique.

The advantage of adaptability decreases the probability of false alarm, because the algorithm changes the threshold and adjusts it at the correct level according to the conditions of the network at that moment. Once an alarm is declared, the algorithm starts considering that the next behaviour is normal again.

The interest of this kind of algorithms lies in the improvement of the network management, because they don't need to be reconfigured once they are implemented in a network. They adapt to new situations which couldn't be considered when the algorithm was implemented.

In the other hand, adaptive algorithms require an important amount of stored information. And this is a problem when the network is complex because the stored data is proportional to the size of the network. But this effect can be softened by choosing the correct algorithm.

Furthermore, processing time is another problematic parameter. It depends both on the quantity of data and the complexity of the algorithm. As well as the amount of stored data, the processing time grows proportionally to the size of the network. If the processing time is not short enough, some alarms may not be declared or noticed.

2.3 Kinds of adaptive algorithms

There are a great variety of adaptive techniques developed in order to solve the problem. Here I present three of them each one using different signal processing theories. The technique implemented here uses another approach and it is presented in point 2.3.4. I chose this algorithm because I thought it was easier to implement and to understand. At the end of this report the audience will notice that the results obtained are not as satisfactory as I thought at the beginning.

For the moment, let's see some ways to face the problem. Since the idea of adaptive anomaly detection, several algorithms have been developed as we can see briefly with the following examples. Each one has specific advantages, and the choice depends on the necessities of the network system.

2.3.1 Maximum entropy method

This proposed algorithm has been taken from the work of Yu Gu, Andrew McCallum and Don Towsley (6). They compare the information obtained from the network data with a baseline distribution. This is made by using the Maximum entropy method. This technique presents a new advantage, these algorithms can realize about the difference between abrupt and slow changes. Thus, the type of anomaly can be distinguished.

2.3.2 Network anomalies using sketch subspaces

Xin Li, Fang Bian, Mark Crovella, Christophe Diot, Ramesh Govindan, Gianluca Lannaccone and Anukool Lakhina proposed this technique (7). In this work the idea is to compare traffic sketches with a subspace method. They say that the accuracy in anomaly detection using this technique is high. The technique detects which flows are the causes of the anomaly besides the IP addresses which origin it.

2.3.3 ARP-based Anomaly Detection Algorithm Using Hidden Markov Model

There are methods specified for different kinds of traffic, as this one that uses the information of the ARP traffic and applies a new algorithm focusing on Hidden Markov Model. The development of this method can be followed in the paper "An ARP-based Anomaly Detection Algorithm Using Hidden Markov Model in Enterprise Networks" (8).

2.3.4 Anomaly detection in IP networks using AR-models.

This technique bases its idea in detecting abrupt changes over predefined MIB variables which represents the state in the network element. Further information about this algorithm can be found in the paper “Anomaly detection in IP networks” (1). This is the algorithm chose for my design. It will be explained in more detail in Section 3.

3 Algorithm Development

This section contains the algorithm description as well as the environment where it should work. First point 3.1 describes the algorithm and the theoretical characteristics

3.1 Proposed algorithm and technical description

There are several ways to solve the problem with adaptive algorithms, as we saw briefly in Section 2.3. But for the moment, we focus in a selected algorithm that will be developed here. As it was explained the algorithm proposed has been taken from the results of Marina Thottan and Chuanyi Ji. The work is titled: “Anomaly detection in IP networks”.

I chose the mentioned technique because I found easy to implement it in the environment of the university. The paper was easily understandable for me, and I saw that the system can be scaled to the size of the network. I wanted to develop this technique, and prove that it works in our environment as well as in the environment proposed in the paper.

The writers explain that there are some ways to confront the problem. As they say: *“Researchers have approached this problem using various techniques such as artificial intelligence, machine learning, and state machine modeling”* (1). They classify network anomalies in two groups:

1. Problems related with network failures and performance problems. File server failures, paging across the network, broadcast storms, babbling node, and transient congestion are performance anomalies.
2. Security-related problems. Denial of service attacks and network intrusions belongs to the second group.

In advance, my intention was to detect all of these anomalies, and to specify where they take place. Unfortunately, the test environment only allows me to detect anomalies in one network element, so I only can detect the anomalies in this place. For future developers shouldn’t be difficult to extrapolate this work to an entire network, and limit the failure to the area controlled by the network element which produces an alarm.

In order to do this I have followed the steps proposed in the paper which will be explained in Section 3.1. For the data acquisition, I propose an alternative process, and I will decide if it is better or not that the one proposed in the paper.

3.1.1 Application environment

3.1.1.1 Typical architecture.

In this section, I explain the application environment of the algorithm selected, which is the common architecture of it. Let’s consider the following structure.

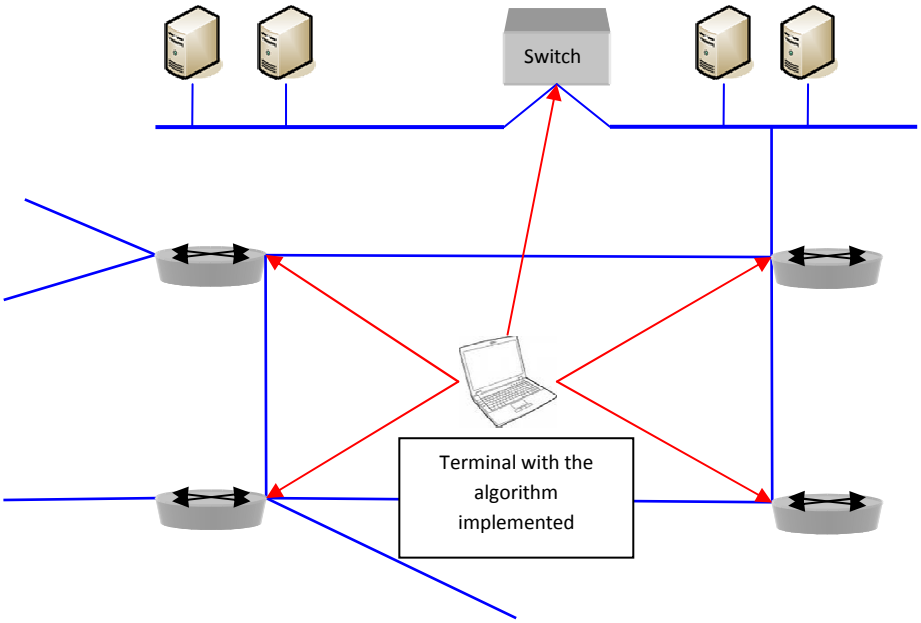


Figure 1

Usually these network elements are routers dividing the network in sub networks and switches separating domains in a LAN. The application of the algorithm in each network element brings as result if the domain of the element is suffering an anomaly or not.

Using the algorithm in all important network elements, we are able to limit the area of the failure in order to easily solve it.

3.1.1.2 Algorithm adaptive law.

The selected algorithm bases the idea of learning, in the past data store. We define the algorithm to present the result each window time (we'll see what is this later), but the information given by a predefined number of previous windows is stored and affects to the value of the current result.

The idea of the algorithm consists on check the behaviour of the network during a predefined period. When the behaviour changes, that means when an abrupt change occurs, the algorithm gives a false alarm. The next steps, the algorithm tries to adapt to this new situation. As the time goes on it can be consider a new normal behaviour of the network.

For example, if a server is disconnected, it will be detected, but if nobody solves the problem, the network traffic will be consider normal when the precise time has passed. It is very important to configure correctly the algorithm the first time, because the future behaviour of it will be affected.

3.1.1.3 Software used and its application.

For implementation I'm using some tools that will be explained in this section. First of all, the environment runs in Linux Machines. I have decided to use Linux because the tools able for working in this project are available in Linux environment and they are open-source.

I design this system basing some points in the developments of my partner Laura Herranz. She is designing a system which is able to detect new attacks to the network of the university. The idea is to implement two adaptive tools able to identify the problems in the network. Because of that we share the laptops, and we have worked together in this part of the project.

1. **Linux platform:** We had too many problems with Linux and libraries installation. Finally my laptop works with UBUNTU 8.04, and the laptop which belongs to Laura is configured in the same situation.
2. **SNMP:** I use a Linux package for the data acquisition. As the rest of Linux tools this package is open-software and it let me to get the MIB variables from the network element. The functions given by this package allow obtaining the data in different ways. I use the function *snmpdelta*. The parameters needed are the IP address, the time between requests and the MIB variables which are requested. This information is stored in a temporary file with a table format; each variable is stored in one column.
3. **Packet Generator:** It is another Linux application. It is used to generate traffic and simulate a normal situation in the net. It will be useful for the test part of the design in order to obtain a good behaviour in the network, because the traffic in the test network is not very stable. The generator is a Linux command which generates flow without information from one machine to other crossing the router.
4. **MATLAB 7.0:** This is used for processing the information obtained with SNMP and adapt it in order to facilitate the processing of the data. The algorithm is implemented in MATLAB. MATLAB is the main program used in this work. Its functions are processing information, algorithm application, and results launch.
5. **Attacks generator:** This package provided by my partner Laura Herranz, simulates attacks to the network we want to check. It sends a burst of attacks in a short time, and it is used to simulate failures over the network and check the correct behaviour of the algorithm.

Linux distribution was obtained for free from the reference (9). SNMP and PacketGenerator are obtained for free also, because they are open-source software.

In the case of MATLAB, it was downloaded with the university license.

3.1.2 Implementation

3.1.2.1 Starting

The algorithm is implemented over MATLAB and the data acquisition over shell-script. The script programmed is run first. It executes the *snmpdelta* function in order to obtain the data in a file. The data is stored in a temporary file and when the data acquisition finish, MATLAB starts processing the data. This communication between MATLAB and shell-script is made by the use of system-signals.

At the beginning MATLAB is executed from the shell-script, and then the shell-script starts with the data acquisition. MATLAB sends a WAIT-signal to itself. When the file is prepared for MATLAB, the shell-script sends a CONT signal to the MATLAB process, and then it starts with the algorithm. Once the algorithm has finished with the data, MATLAB resend the WAIT-signal to itself, and the iteration starts again.

3.1.2.2 Data acquisition

In the paper is proposed a method for data acquisition. They use the application layer protocol SNMP (10) (11) in order to obtain the MIB (12)variables of each element in the network. These variables contain the information needed by the algorithm proposed. MIB variables for routers[1] may be: number of packets received from all the interfaces (*ipIR*), number of packets correctly delivered to higher layers as this node was the last destination (*ipID*), and number of packets successfully sent from a higher layer of this router (*ipOR*).

In my design, I first tried something different, but, finally, I used SNMP. I wanted to use the network protocol Net Flow (13). It is property of Cisco Systems and it is implemented in the routers of the university.

The acquisition of data in this case shall be done obtaining the information of the network flows which cross the router. The management of this flow information is carried out with the program Flow Tools (14) (15).

Flow Tools is open-source software, and it runs under UNIX platforms. In my case the OS (operation system) used is Linux UBUNTU 8.04. The laptop receives the Net Flow information from a router and work with it using the Flow Tools software.

Because the information received is not the same with Net Flow than with SNMP, I had to process it before using the algorithm. While I was doing this, I realized that the overload in processing time was huge. Thus I decided to reject this solution, and follow the one proposed in the paper.

In order to get the MIB variables from the router, I needed to install the SNMP package for Linux. With some of these functions I could ask the router for the three MIB variables needed in my design. The data acquisition works in the following way:

- Be T_S the sample period, the router is asked by the data acquisition routine each T_S, and the routine receives the incremental value of the MIB variables since the last request.
- The script which runs this routine, writes the data in a temporary file, and stores the information received during a window time, T_W. The window time will be explained in next section.
- Then the temporary file is copied in other file, from which data will be processed. Now, the script can overwrite again the temporary file with the new data to store.

This procedure is necessary because otherwise data acquisition routine and data processing routine could access to the data at the same time, causing lost of data.

Eventually the file format is composed by *n* columns (being *n* the number of MIB variables), and r rows (r is the number of samples inside a window period).

[1] These MIB proposed are for routers only. In case of switches, the writers propose to run a simple Principle Component Analysis in order to get the MIB variables that fix better the problem.

3.1.2.3 Data processing

The algorithm for anomaly detection bases itself in processing signal theory. Our desire is to get a *healthy function* (f_h) for each router (I take variables only from routers, not from switches), which can be compared to decide if the network suffers an anomaly or not in the mentioned router. This *healthy function* depends on the variables that we have chosen to characterize the behaviour of the network (e.g. number of packets received in a specific port, number of packets transferred to upper levels, etcetera.). As the audience can read in the reference proposed above, the *healthy function* is defined as:

$$f_h(\vec{\varphi}(t)) = \vec{\varphi}(t) \cdot A \cdot \vec{\varphi}(t)'$$

Where A is a $M \times M$ matrix, meaning M the number of variables chosen, and $\vec{\varphi}(t)$ the *abnormality vector* $1 \times M$-sized. A-matrix is the matrix of a linear operator, thus it has special requirements that shall be defined later on. In the next section it is specified the process to obtain the *abnormality vector*.

Matrix A is design to be symmetric because it needs to be a linear operator. Several eigenvectors that compose it are associated with the anomalies of the network. And the minimum and maximum eigenvalues associated to them are λ_N and λ_M respectively.

An alarm shall activate in the instant:

$$t_a = inf\{t: f_h(\vec{\varphi}(t)) \geq \lambda_N\}$$

That means that an anomaly has occurred. But it will be explained in more detail in the following lines.

3.1.2.3.1 Abnormality vector.

In order to calculate the *abnormality vector* we need to obtain information from the network. First of all the information is obtained from each router, and separated in different MIB variables (e.g. number of packets received from a specific port, or number of packets transferred to upper levels, etcetera.). The time between two information requests is called sampling period T_S[2]. We consider a number of samples contained in a window period T_W[3] for each iteration of the algorithm.

In each sampling period the algorithm request the network elements for the MIB variables. The objective is to detect the abrupt changes in the MIB variables. Let's focus in one network element in order to make easy to understand the algorithm.

The algorithm is applied for each window as it can be seen in the next Figure 2.

[2] T_S is defined around 15 seconds.
[3] This period is 5 minutes. T_S and T_W are design parameters, and should be fixed in order to obtain the best behavior of the algorithm.

MIB variable “n” (e.g. number of packets received from a specific port):

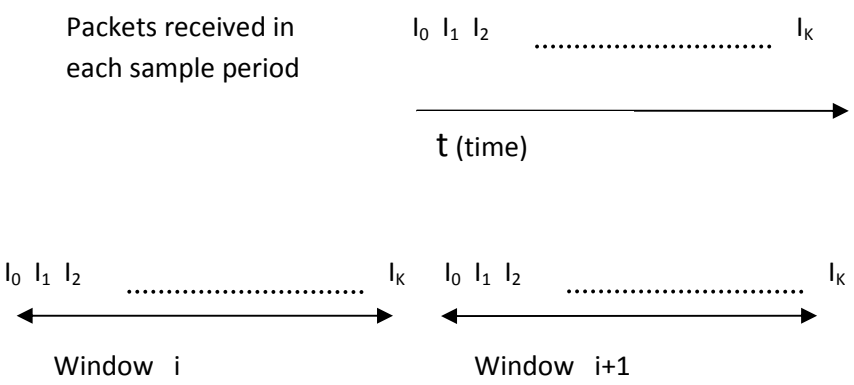


Figure 2

Once the algorithm has enough samples for working (we have complete a whole T_W period), it processes each variable separately. It considers the samples of each variable as an AR-Model (16).

The AR-Model assumes that the data are correlated in time, so abrupt changes can be detected with past information. In order to obtain an accurate approach, the order of the AR-Model must be short, in fact the writers propose a model with order one. Order is also called degree and is represented with the variable *p*. The rule of an AR-Model is presented in the next formula, but we are only interested in the variance obtained from the model as we will see.

$$R_i = c + \sum_{t=1}^{p} a_t \cdot R_{i-t} + \varepsilon_i$$

The constant *c* is null in most of the cases as the audience can see in the reference (16). Here are presented another constants called a_t. Finally the random variable ε_i is a Gaussian error term, which parameters are null mean and variance σ^2.

Let be the series of samples shown in the Figure 3, considering them as an AR-Model, we get the parameters mean and variance related with this model.

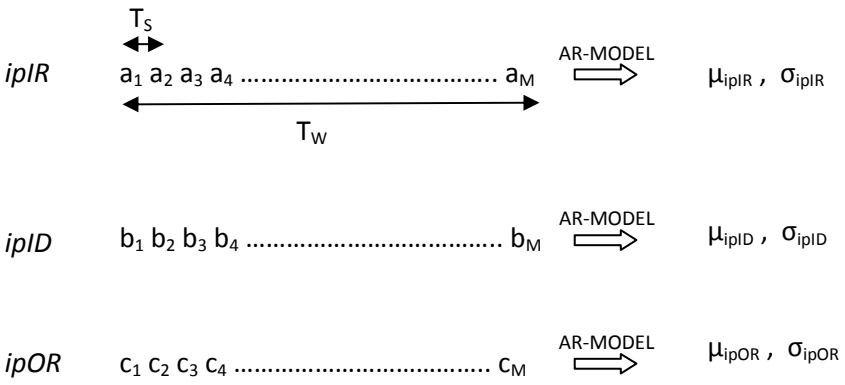


Figure 3

With this process we get the variance associated to each MIB variable, and in the whole window. This variance is called in the reference as pooled variance. But the algorithm necessitates two parameters else. Writers divide the window in two groups.

Thus window samples are separated in two groups: learning (L(t)) and test party (S(t)); and the variances of each one are obtained considering each group as a different AR-Model. The first "N_L" samples are aggregated to the learning party, and the rest samples goes to the test. We need to calculate the statistics of each group, obtaining the variance, as we did for the whole window. Figure 4 shows the parameters N_L and N_S.

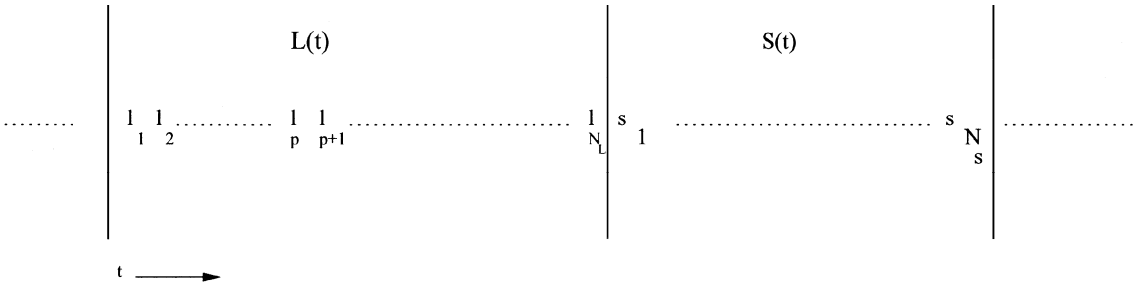


Figure 4

Where N_S and N_T are the number of samples in the first group (*learning*) and in the second (*training*).

The purpose is to obtain the component of the *abnormality indicator* associated to each MIB. In order to get this result we use the GLRT (generalized likelihood ratio test) algorithm. The data are tested using the GLR (generalized likelihood ratio).

The likelihood ratio or *abnormality indicator* depends on the variances calculated previously, as can be noticed in next formula:

$$\varphi = \frac{{\sigma_L}^{-\hat{N}_L}{\sigma_T}^{-\hat{N}_T}}{{\sigma_L}^{-\hat{N}_L}{\sigma_T}^{-\hat{N}_T} + {\sigma_P}^{-(\hat{N}_L+\hat{N}_T)}} \gtrless 1$$

$$\grave{N}_L = N_L - p$$

$$\grave{N}_T = N_T - p$$

Parameter p is the degree of the AR algorithm. Parameters σ_L and σ_T are the variances obtained considering each group as an AR-Model. Finally the parameter σ_P is the pooled variance of both groups, considering the whole window for the AR-Model.

Once we have the abnormality indicator for each variable, we join them in a single vector which will be called *abnormality vector*, and will be defined as $\vec{\varphi}(t)$. In the case we work with a router we considered three MIB variables, and the *abnormality vector* is defined as:

$$abnormality\ vector = \vec{\varphi}(t) = [\varphi_{ipIR} \quad \varphi_{ipID} \quad \varphi_{ipOR}]$$

For the rest of network elements the size of the abnormality vector is the same as the number of MIB variables used for it.

$$\vec{\varphi}(t) = [\varphi_1 \quad \varphi_2 \quad \dots \quad \varphi_M]$$

Finally for the purpose of the normalization, a new component is added at the end of the abnormality vector, which represents the normal functioning of the network (probability of good behaviour). This component is updated each iteration of the algorithm. Then the component α is calculated in order to fix the next condition:

$$\vec{\varphi}(t) = \alpha[\varphi_1 \quad \varphi_2 \quad \dots \quad \varphi_M \quad \varphi_0]$$

$$\langle \vec{\varphi}(t), \vec{\varphi}(t) \rangle = 1 \quad \rightarrow \quad \alpha$$

Parameter α is the normalization constant.

3.1.2.3.2 A-matrix.

As is defined in the paper, the *A-matrix* is formed with the following law but now, the φ_0 component is not considered because it is used only for normalization purpose:

$$A(i,j) = \langle \varphi_i(t), \varphi_j(t) \rangle = \frac{1}{T} \left| \sum_{t=1}^{T} \varphi_i(t) \varphi_j(t) \right|$$

$$A(i,i) = 1 - \sum_{j \neq i} A(i,j)$$

The parameter T is a design parameter, and represents the memory of the system. It is the number of previous abnormality vectors calculated used in the A-matrix definition. By this way, A-matrix contains the information of the network in the last T windows.

Once the matrix is calculated, it is necessary to obtain the eigenvectors and the eigenvalues. Parameters λ_i are the eigenvalues of A-matrix associated with anomalies in the network and $\overrightarrow{\phi_i}$, the eigenvectors related with them. There are some eigenvectors with their eigenvalues associated, that form the *faulty region.*

The *faulty region* is defined as the group of eigenvectors that belongs to a linear combination of the current abnormality vector:

$$\varphi(t) = \sum_{N}^{M} c_i \overrightarrow{\phi_i}$$

Then the faulty region represents every failure state in the network, and depending on the previous linear combination, the abnormality vector is approached at the corresponding state of the network. The coefficients c_i are the weights of each eigenvector and ${c_i}^2$ represent the probability of the network of being in the state i-th.

These parameters are very important because they will be used in order to define the faulty region. If the coefficient (${c_i}^2$ because it is the probability) associated with a pair eigenvector-eigenvalue, is very low, then this eigenvector doesn't belong to the faulty region[4].

In the paper the next result is explained too. This rule gives the *network healthy indicator* in the current window, and helps to decide between alarm and normal behaviour of the network.

$$E(\lambda) = f_h\left(\vec{\phi}(t)\right) = \sum_{i=1}^{M} {c_i}^2 \lambda_i$$

The alarm is activated when $E(\lambda)$ is upper than the minimum eigenvalue which belongs to the faulty region:

$$E(\lambda) > \min_{\lambda_i \in \{\lambda_N, \lambda_{N+1}, \ldots \lambda_M\}} (\lambda_i)$$

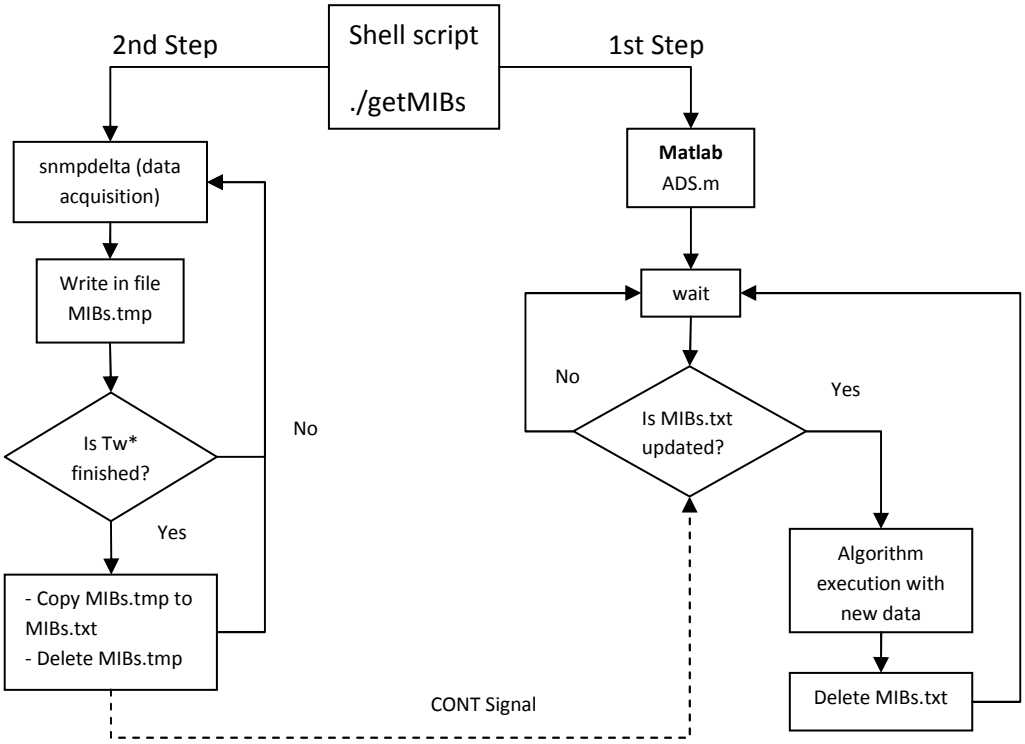
As summary of this, I would like to emphasize how the algorithm adapts itself to the new situation. Each window, the algorithm takes data from the network, and when a window is complete, it processes the data in the explained way. Then the next iteration starts. The processing of the data network has to consider the past information. The past information of the network is stored in the A-matrix through the T past abnormality vectors. And abrupt changes detection is done by using an AR-Model of order 1, which let us detect changes occurred in a sample period. Other component that helps to consider the past information is the probability of good behaviour of the network. This parameter, as audience can remember, is added to the abnormality vector at the end, and represents the state of good functioning of the network. All of these steps in the algorithms aid to introduce the past information in the last decision of the system, when an alarm is declared or not.

3.1.2.4 Flow diagrams

The process explained in point 3.1.2.1, is represented in next flow diagram. It helps to understand the communication between shell-script and MATLAB.

Figure 5 shows the flow diagram which represents this point.

[4] The parameter ${c_i}^2$ is contained in the range [0-1], thus in my design, the threshold over which the eigenvector is consider belonging to the faulty region, is set to 0'1.



* Tw is the window time; it was explained in the algorithm description

Figure 5

Some try-catch blocks can be found while running the software. These blocks allow the system to finish correctly the execution when an error occurs.

As we can see in the previous picture the communication between MATLAB and shell-scritp has two ways of action. First one is the signalling communication. Using signals both threads can make use of the same files and in the correct order. So they will not destroy the work of the opposite thread. The second way of communication and the most important one is the data sending. Shell-script obtains the data from the router, write them in a file, and once the data acquisition is complete, it writes a new file which will be read by MATLAB at the appropriate time.

Now I focus in the flow diagram which represents the algorithm implemented. There is possible to observe the steps followed in the previous point where the algorithm was explained.

Figure 6 shows the flow diagram which helps to understand how the algorithm works.

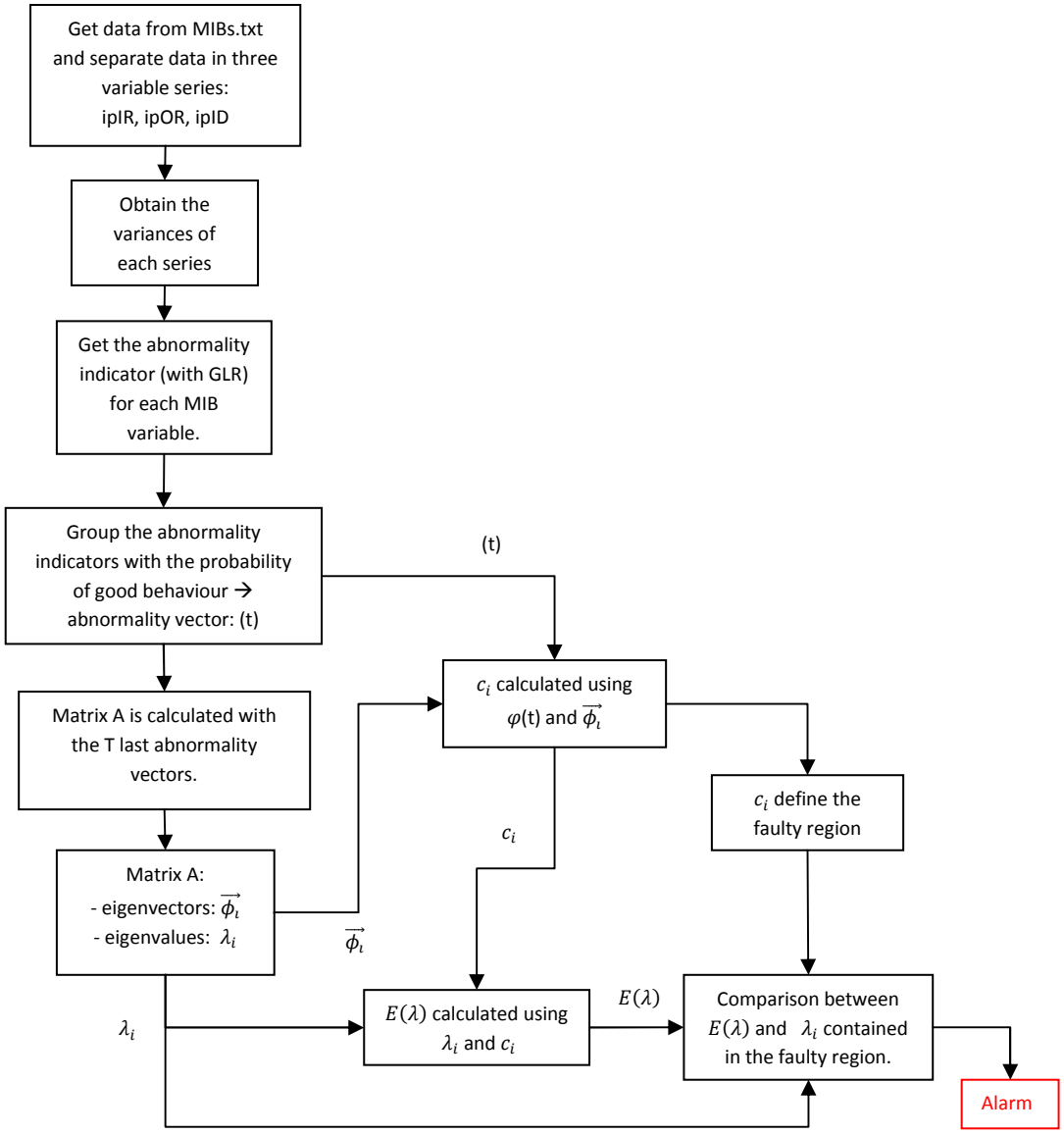


Figure 6

3.2 Problems found

During the implementation of this algorithm I have solved many problems that affected directly the behaviour of the system. They have been technical problems and concept problems.

3.2.1 Understanding the algorithm.

One of the main parts of the work consists in a good understanding of the algorithm. During this task, I found some dark points. The bibliography helped me too many times. However, there were some points that couldn't be solved neither with the bibliography nor Internet. In those cases, I had to contact the writers. They helped me with my doubts.

3.2.2 System installation. Problems with the platform and package installation.

Choose the correct platform was important, because MATLAB over Linux doesn't run very well in some distributions. I tried a few distributions but finally I decided to use Ubuntu after check

in Internet that it is one of the most complete and stable distribution for Linux. After the decision, I didn't notice problems with MATLAB anymore.

Even with the rest of packages I installed (Flow Tools, SNMP) I didn't have any problem of compatibility.

3.2.3 Mounting the workstation.

There were some problems not very important, during the installation of the workstation. I needed to connect my laptop to the router which I was going to use for the software design and test environment. For the correct functioning of the SNMP protocol, I was given some permission over the router. After that SNMP could run correctly.

I realized that every connection to network elements should have this permission, if we are thinking in controlling the entire network.

3.2.4 Understanding Flow-Tools, SNMP package and shell-script.

All of this software has a complete and useful bibliography. I had many problems trying to understand how these tools work, but the access to the bibliography gave me a clear idea about them.

3.2.5 Data acquisition: Flow-Tools or SNMP.

As it was explained, the first idea was to implement the data acquisition with a tool able to manage the flows that runs over the network. I thought that with this capability I could identify not only the network element which fails, but also the original flow which causes it.

Going deeply in the study of the algorithm, I realized that my intention wasn't possible, because the algorithm was designed for detecting the anomalies in the network element. The charge of work caused by a design like this was huge, and I decided to follow the paper even in this part.

For further developments it could be possible to use a flow analyzer like Flow Tools. We would need to collect all the flows which cross over the network element and then process them in order to obtain similar information than MIB variables give. Besides, it doesn't seem to improve the behaviour of the algorithm. This processing charge, the waste of time trying to adapt this solution to the algorithm and the possibility of a small improvement in the efficiency, decided me to change my mind and follow the specifications of the paper.

3.2.6 Implementation in MATLAB.

I didn't found too many problems in the implementation of the algorithm. I chose MATLAB because it contains functions for all the formulas and algorithms I needed to use. I had to decide between the different MATLAB's AR functions. Finally I chose one of them, the most common, imaging that the results would not vary too much.

Actually there was a problem with the MATLAB implementation. When I obtain the variances, sometimes the value obtained is indeterminate form. In these cases I was force to bring any value to the software in order to make it work properly. I defined all the indeterminate forms that the algorithm could bring with their associated real value. Then when the result is indeterminate form the software assign to the result the associated real value.

Figure 7 shows the correct values of abnormality indicator in case of indeterminate form:

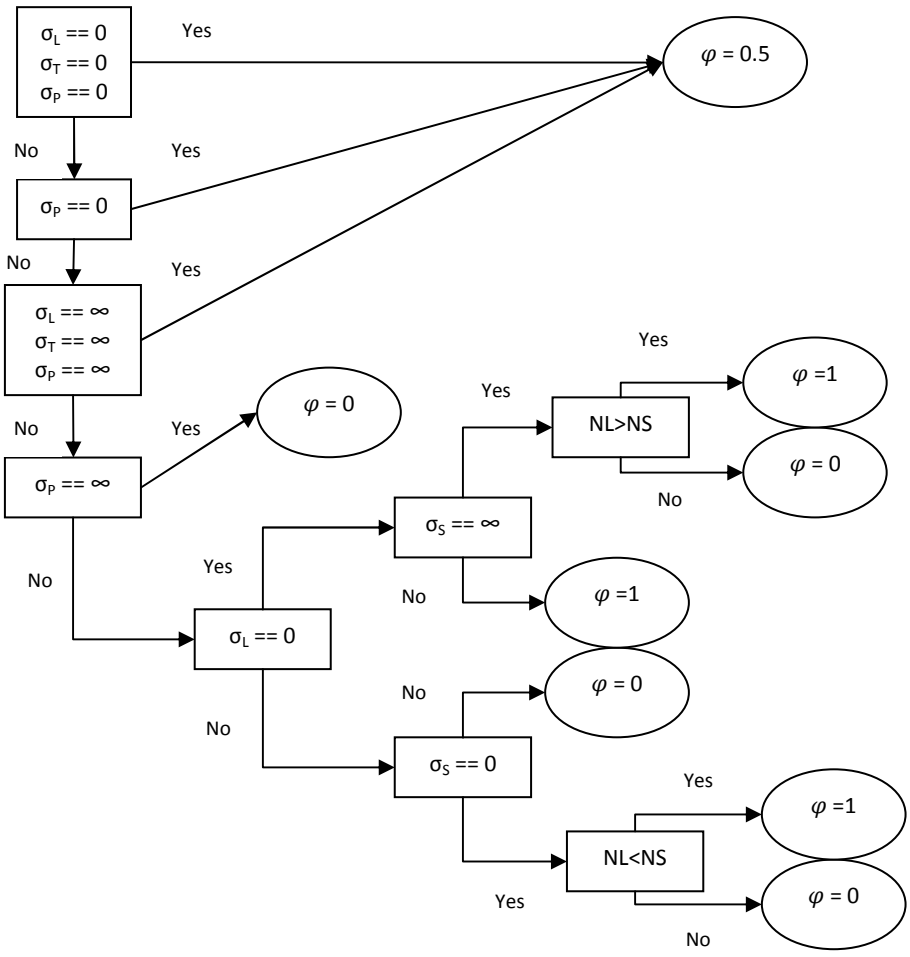


Figure 7

3.2.7 Signal communication between MATLAB and shell-script.

I spent too much time trying to solve problem with the synchronization between MATLAB and shell-script. I got too many problems trying to stop MATLAB, while it hadn't available data. Finally I found the MATLAB function which lets it to send system-commands. I used this function to send a WAIT-signal from MATLAB to itself, while the CONT-signal was send by the shell-script.[5]

This problem was important because if MATLAB and shell-script are not synchronized, the system finally fails. Depending on the T_W and the time spent in the algorithm, there will be a moment when MATLAB tries to access to a file that doesn't exist and finishes its execution, while the script continues with the data acquisition.

3.2.8 MATLAB execution from shell-script. Configuration file.

When shell-script executes MATLAB for the first time, introduce arguments on it is impossible. I wanted to introduce some parameters by the command-line, such parameters are AR-Model

[5] This explanation was explained in section 3.1.2.1 and graphically in section 3.1.2.4.

degree, or number of samples used for the learning window. But I found impossible to send them by arguments to MATLAB, so I decided to create a configuration file at the beginning. This file is created by the shell-script, with the data it receives by arguments. Then, when MATLAB starts the first thing that does is to read the configuration file, and obtain the values of the parameters.

3.2.9 Read the data from MATLAB.

The function which lets MATLAB to read correctly the data from the file is called *textscan*. This function is very useful because it allows reading a file and interpreting the spaces as cells separations in a table.

3.2.10 Exceptions caught.

When something is not working, every unsuspected error is caught by an Exception block. This block closes the shell-script process which is obtaining the data. It assures that everything is closed in the correct way. I had too many problems before implementing this block in MATLAB. But it was impossible to implement the block in the shell-script; I couldn't find a function that lent me catch the exceptions.

3.2.11 Probability of good behaviour. Solution proposed.

During the algorithm implementation, I found another important problem. The writers didn't specify how to implement the component *probability of good behaviour*. So then I decided to act by myself again.

I start the algorithm considering this parameter with a value like 0'9. Then each i_{th} iteration I apply the next formula:

$$p_i = p_{i-1}\frac{i-1}{i} + \frac{1}{i}$$

4 Tests and Results

The goal would be to work in a normal network and make one or several servers to fail, as writers of the reference paper make. But due to the situation where my work is carried out, the only thing I can do is to attack the network and observe the failures produced (which is not the main purpose of the algorithm).

The paper brings some examples of application, and different environments with respective attacks. But the authors don't present a unique configuration. They give the idea of the algorithm but don't present their own parameter configuration for their tests.

This task is one of the steps that I develop in my design. The algorithm must be adapted to the conditions of the network in use. This should be a previous work in each implementation. I give the idea of how these parameters affect to the algorithm behaviour. The purpose is to bring a brief developer's guide which let the designer to limit and facilitate the configuration time.

This point is divided in two parts: Test environment and Results. In the first part, the environment used to test the algorithm is provided. It is also discuss if the environment proposed is suitable to represent the behaviour of the algorithm in other situations.

The Results section presents the results of the tests executed. The audience can find there the study of the best parameters, and the behaviour of the algorithm already configured with these parameters. Besides, it is possible to find there the analysis and comparison with other solutions.

4.1 Tests environment

The development of the work takes place in the University of Stuttgart, and it has been carried out with the help of the people working in RUS department.

I was offered an environment where I could develop the work. I received connectivity to the main router in the network of Daidalos[6], and I carried out all my tests in this network.

For further information about Daidalos, visit the reference (17).

I was given a laptop in which I installed all the necessary tools and I connected it to the main router of this network. My laptop became a node of the network, and I could search the information that I needed for my algorithm. The traffic generated over the network by my algorithm is negligible.

Figure 8 shows the connections and configuration of all the elements participants in the test bench.

[6] Daidalos is a project carried out by several universities and companies. The University of Stuttgart belongs to this group of universities, and it has a network for this programme.

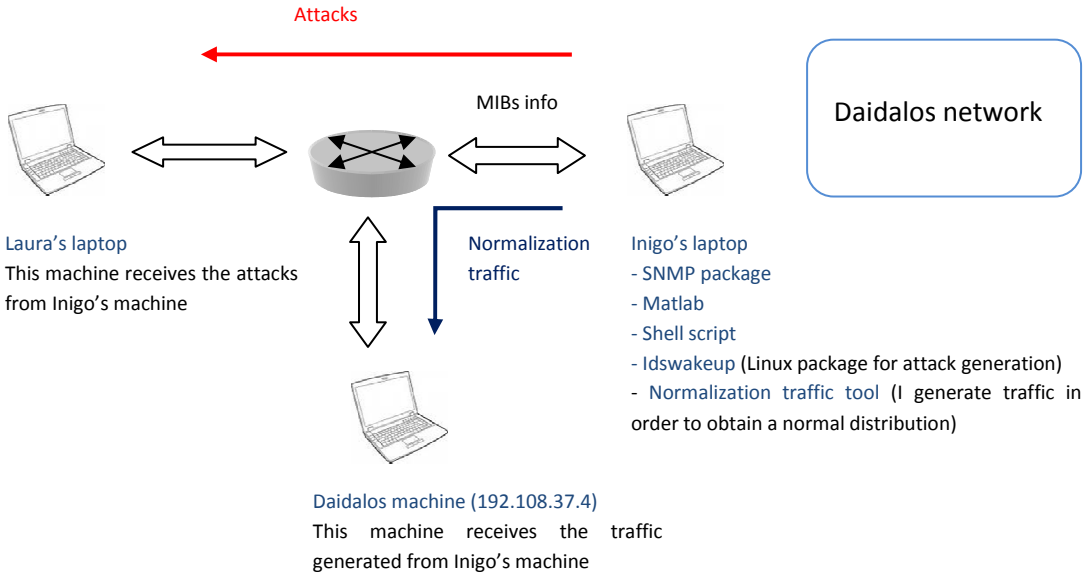


Figure 8

As we can see in the Figure, the tools used were described in previous sections. Furthermore, there are other functions needed to simulate special situation on the network. For example, the Linux package *idswakeup* is used to simulate malefic attacks to the network. In other tests I create traffic in order to normalize the network traffic. In this situation the traffic produced by my system is not negligible, but allows simulating the traffic present in a "normal" network (recall that the router is used in the Daidalos network, a network for tests).

The result of the tests consists in two files:

- First one stores the alarms given by the system. It is necessary for presenting them in the graphics shown in Section 4.
- The second file stores the full response of the system, including abnormality vector, A-matrix, eigenvalues, alarm and the probability of good behaviour besides of other parameters. This file let me understand which has been the progress of the algorithm and obtain conclusions for each situation.

Below, both files are showed:

File Alarms

24
28
52
71
77
87
97
98
100
102
103
104
105
108
110
112
113
123
125
126
128
129
131
133
136

Figure 9

This file keeps the iteration when an alarm was declared. In this example, the iteration is with five minutes of length. The complete test had twelve hours of duration.

For the data presentation, I show a graph with time in the abscissa's axis and alarm declared in the ordinate's axis. I only calculate the time of an alarm declaration by multiplying the iteration number by the iteration length.

File Results

```
Iteration 6
A =
    0.6434    0.1374    0.2192
    0.1374    0.7025    0.1601
    0.2192    0.1601    0.6208
ci =
    0.0664
   -0.0208
   -0.8685
eigenvalues =
    0.4102
    0.5565
    1.0000
E_lambda =
    0.7564
Warning: No alarm:6
> In ADS at 86
p_anom =
    0.9167

Iteration 7
A =
    0.6283    0.1545    0.2173
    0.1545    0.6558    0.1898
    0.2173    0.1898    0.5929
ci =
   -0.2230
   -0.7120
    0.2143
eigenvalues =
    0.3839
    0.4930
    1.0000
E_lambda =
    0.3149
Warning: No alarm:7
> In ADS at 86
p_anom =
    0.9286

Iteration 8
A =
    0.6434    0.1600    0.1966
    0.1600    0.6608    0.1792
    0.1966    0.1792    0.6241
ci =
   -0.8089
    0.0491
    0.2996
eigenvalues =
    0.4324
    0.4959
    1.0000
E_lambda =
    0.3739
Warning: No alarm:8
> In ADS at 86
p_anom =
    0.9375
```

Figure 10

This second example file contains the results of three iterations of the algorithm. The first parameter obtained is the *A-matrix*, followed by the c_i components, which allows the algorithm to decide which eigenvectors belong to the *faulty region*. The third parameter showed is the *network healthy indicator* which is compared with the eigenvalues of the eigenvectors in the *faulty region*. The result of the comparison is presented in red, indicating if the alarm is declared or not. As we can see, there are not alarms declared in this file, that's why the *probability of good behaviour* in the network is increasing each time.

4.1.1 Tests structure

In section 4.2 we develop the tests. They are divided in four groups. The three first groups are those formed by the tests run in order to obtain the best parameter configuration of the algorithm (Tests 1 to 22). The fourth group are the results of the application of the algorithm with the best parameters configuration.

First group of test (Tests 1 to 8) have the same parameter configuration as second group of test (Tests 9 to 16) respectively, but in different environmental situations. Both groups helped me to specify with more accuracy the parameter configuration of the third group (Tests 17 to 22). In this case the parameters configurations at the beginning are:

	AR-Model order (p)	Percentage of learning samples $\left(\frac{N_L}{N_L+N_S}\right)$	Initial probability of good behaviour (p_anom)	Memory of the algorithm (T)	Sampling period in seconds (T_s)	Window time in minutes (T_w)
Tests 1 & 9	1	80	0.9	6	15	5
Tests 2 & 10	1	80	0.9	2	15	5
Tests 3 & 11	1	80	0.9	15	15	5
Tests 4 & 12	1	80	0.9	6	3	2
Tests 5 & 13	1	80	0.9	6	45	20
Tests 6 & 14	1	20	0.9	6	15	5
Tests 7 & 15	1	40	0.9	6	15	5
Tests 8 & 16	1	80	0.5	6	15	5

Table 1

But I needed to run these tests in two different situations. At the beginning the tests were done supposing that there weren't normalization traffic over the network and attacks (Tests 1 to 8) and they are scanned during a time of twelve hours. Then in the second situation the tests were executed generating a traffic flow which crossed the router and attacking from one network interface of the router to another (Tests 9 to 16), and they are executed during 6 hours. For the purpose of normalization I use the command:

cat /dev/zero | ssh inigo@192.108.37.4 cat - > /dev/null&

This command generates traffic from the machine 192.108.37.4 (connected to the router of Daidalos) to the machine where the command is executed (this machine must be connected to a different interface of the router). This traffic doesn't carry any information.

We need to define the next parameter configuration for the third group of tests. This configuration will be shown in next section basing in the results of two first groups.

Once we observe the results with these parameters we decide which values are the best for each one. Tests 1 to 8 suppose that the best case is when the probability of alarm is very small, because these tests are executed under the supposition of no attacks in the network. These tests help to understand which parameters are the most important. Tests 9 to 16 specify which values are the best for solving the problem, because they are approaching better to the final solution (normalization flow and attacks or failures). Then the next group of tests try to fix better the problem approaching more to the real solution. Finally the fourth group, is the application of the algorithm with the best parameters and the results are discussed.

The tests run for this purpose are[7] those executed in normal conditions and there are some attacks applied. Comparing the time response for each alarm, I calculate a mean for this time, and this result is compared with the results given in the paper.

With these results is possible to detect if the parameters defined in the previous section are the best. These results help to identify if the algorithm is giving the solution we were waiting for.

4.1.2 Analysis of the test environment

The main question of this design is if the test environment proposed can represent a general network environment. The answer is yes in most of the cases.

When we manage a network with several routers, we need to apply the algorithm to every router, and obtain a result for it. But each application is independent of the application over other routers, so we can manage several routers at the same time.

If we would like to obtain all the alarms of the routers at the same time, the algorithm should check the whole network during the window period (e.g. the algorithm needs 30 seconds to obtain a decision from a network element, the window time is 5 min, then we can only manage 10 network elements at most). This is one thing to consider if we want to extend the application of the algorithm to other network elements.

Furthermore, we are developing the tests over a router. But what happen with the rest of network elements like switches? In this case, we shouldn't use the same MIB variables, and we'll possibly need to reconfigure and test the algorithm again in order to implement it.

4.2 Results

4.2.1 Test

In this section all the tests run over the system are presented, and the alarms activated during the test period are showed in a graph. Furthermore, some deductions over the values observed in the results file are given.

4.2.1.1 Best parameters decision. Step 1.

After first eight tests I describe which the influence of the parameters on the algorithm is. For each test, the result is presented in a graphic. The probability of alarm and the probability of good behaviour[8] are given In order to make easy to understand the result.

[7] All of them are carried out with the decision of the best parameters made in the previous tests.

4.2.1.1.1 Test 1

	p	% of learning	*p_anom*	*T*	T_s	T_w
Test 1	1	80	0.9	6	15	5

Table 2

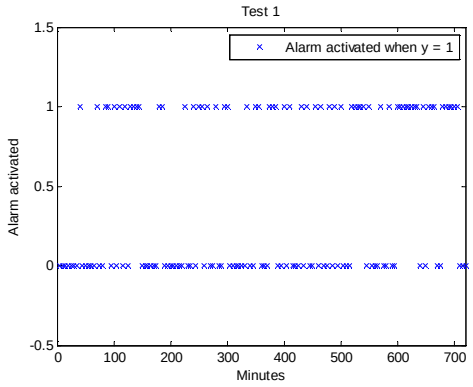


Figure 11

$$Probability\ of\ Alarm = \ 0.4565$$

$$Probability\ good\ behaviour = \ 0.5528$$

4.2.1.1.2 Test 2

	p	% of learning	*p_anom*	*T*	T_s	T_w
Test 2	1	80	0.9	2	15	5

Table 3

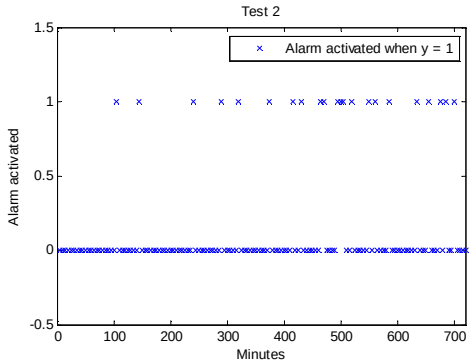


Figure 12

$$Probability\ of\ Alarm = \ 0.1620$$

$$Probability\ good\ behaviour = \ 0.8373$$

[8] The probability of good behavior is the probability calculated by the algorithm. It doesn't know if his alarm is false; thus this alarm will be considered as a failure for the future probability.

4.2.1.1.3 Test 3

	p	% of learning	*p_anom*	*T*	T_s	T_w
Test 3	1	80	0.9	15	15	5

Table 4

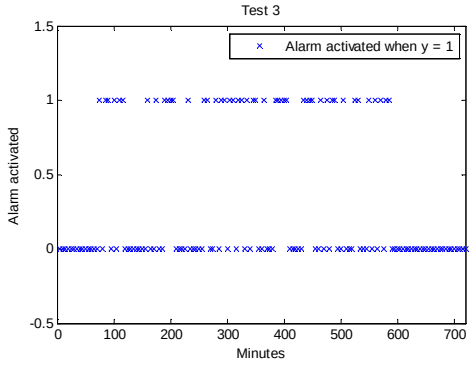


Figure 13

$$Probability\ of\ Alarm = 0.3721$$

$$Probability\ good\ behaviour = 0.5917$$

4.2.1.1.4 Test 4

	p	% of learning	*p_anom*	*T*	T_s	T_w
Test 4	1	80	0.9	6	3	2

Table 5

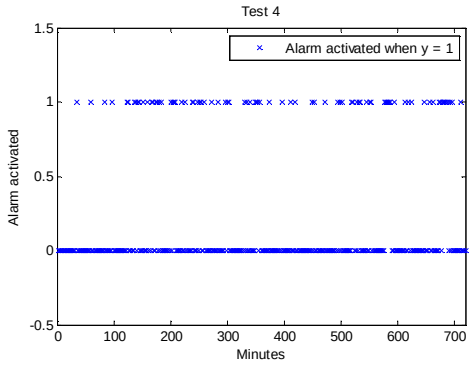


Figure 14

$$Probability\ of\ Alarm = 0.2373$$

$$Probability\ good\ behaviour = 0.5246$$

4.2.1.1.5 Test 5

	p	% of learning	*p_anom*	*T*	T_s	T_w
Test 5	1	80	0.9	6	45	20

Table 6

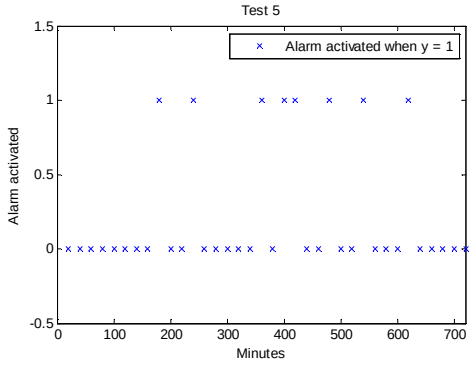


Figure 15

$$Probability\ of\ Alarm = 0.3$$

$$Probability\ good\ behaviour = 0.7286$$

4.2.1.1.6 Test 6

	p	% of learning	*p_anom*	*T*	T_s	T_w
Test 6	1	20	0.9	6	15	5

Table 7

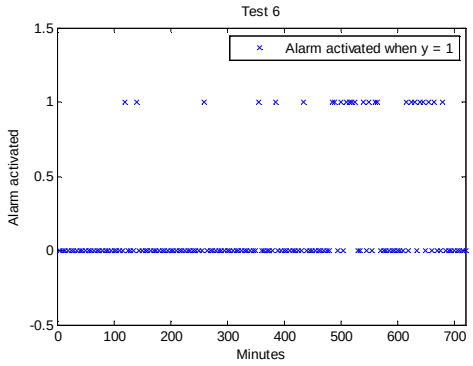


Figure 16

$$Probability\ of\ Alarm = 0.3478$$

$$Probability\ good\ behaviour = 0.8204$$

4.2.1.1.7 Test 7

	p	% of learning	*p_anom*	*T*	T_s	T_w
Test 7	1	40	0.9	6	15	5

Table 8

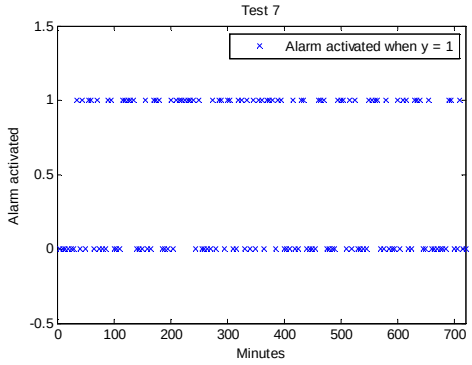


Figure 17

$$Probability\ of\ Alarm = 0.6087$$

$$Probability\ good\ behaviour = 0.5276$$

4.2.1.1.8 Test 8

	p	% of learning	*p_anom*	*T*	T_s	T_w
Test 8	1	80	0.5	6	15	5

Table 9

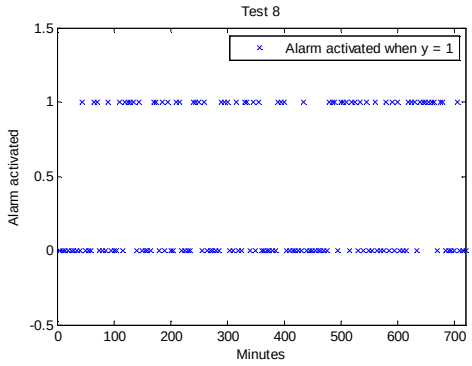


Figure 18

$$Probability\ of\ Alarm = 0.4275$$

$$Probability\ good\ behaviour = 0.5699$$

4.2.1.1.9 Test 9

	p	% of learning	*p_anom*	*T*	T_s	T_w
Test 9	1	80	0.9	6	15	5

Table 10

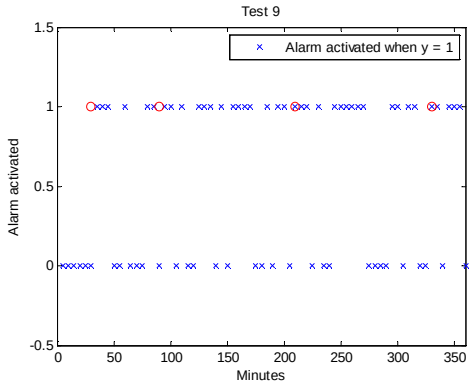


Figure 19

$$Probability\ of\ Alarm = \ 0.7273$$

$$Probability\ good\ behaviour = \ 0.4514$$

4.2.1.1.10 Test 10

	p	% of learning	*p_anom*	*T*	T_s	T_w
Test 10	1	80	0.9	2	15	5

Table 11

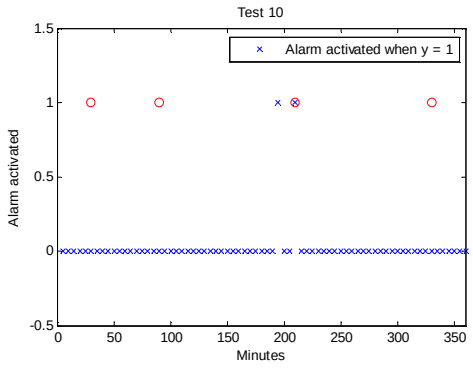


Figure 20

$$Probability\ of\ Alarm = \ 0.0429$$

$$Probability\ good\ behaviour = \ 0.9569$$

4.2.1.1.11 Test 11

	p	% of learning	*p_anom*	*T*	T_s	T_w
Test 11	1	80	0.9	15	15	5

Table 12

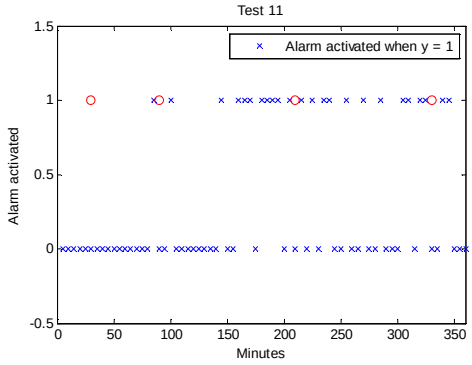


Figure 21

$$Probability\ of\ Alarm = 0.4386$$

$$Probability\ good\ behaviour = 0.6229$$

4.2.1.1.12 Test 12

	p	% of learning	*p_anom*	*T*	T_s	T_w
Test 12	1	80	0.9	6	3	2

Table 13

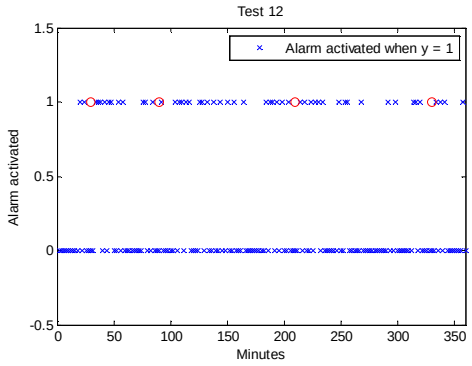


Figure 22

$$Probability\ of\ Alarm = 0.3046$$

$$Probability\ good\ behaviour = 0.7044$$

4.2.1.1.13 Test 13

	p	% of learning	*p_anom*	*T*	T_s	T_w
Test 13	1	80	0.9	6	45	20

Table 14

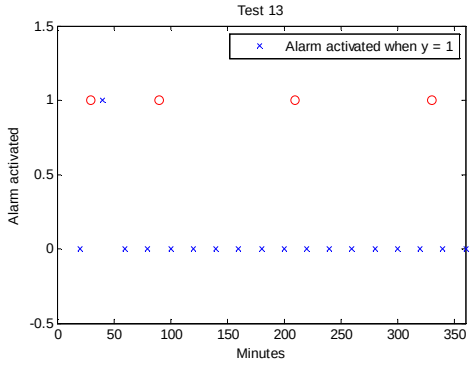


Figure 23

$$Probability\ of\ Alarm = 0.1667$$

$$Probability\ good\ behaviour = 0.8765$$

4.2.1.1.14 Test 14

	p	% of learning	*p_anom*	*T*	T_s	T_w
Test 14	1	20	0.9	6	15	5

Table 15

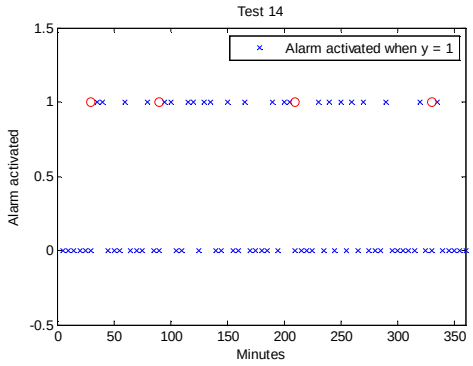


Figure 24

$$Probability\ of\ Alarm = 0.3636$$

$$Probability\ good\ behaviour = 0.65$$

4.2.1.1.15 Test 15

	p	% of learning	*p_anom*	*T*	T_s	T_w
Test 15	1	40	0.9	6	15	5

Table 16

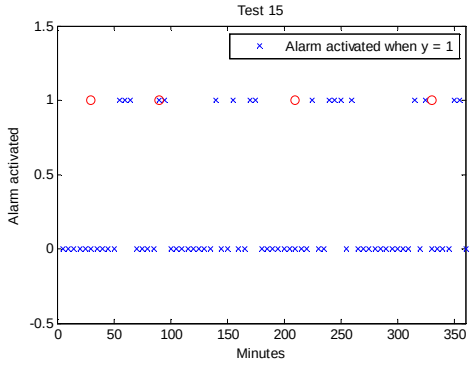


Figure 25

$$Probability\ of\ Alarm = 0.2879$$

$$Probability\ good\ behaviour = 0.7292$$

4.2.1.1.16 Test 16

	p	% of learning	*p_anom*	*T*	T_s	T_w
Test 16	1	80	0.5	6	15	5

Table 17

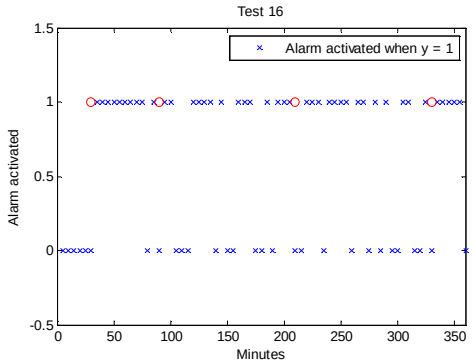


Figure 26

$$Probability\ of\ Alarm = 0.6667$$

$$Probability\ good\ behaviour = 0.3542$$

4.2.1.1.17 Parameters decision for the third group.

The first two groups show us how the algorithm behaves in different situations. Let's see which conclusions are obtained for the next group parameters configurations.

During the first eight tests we look for the algorithm which brings fewer alarms. We suppose that during the execution of these tests there are not attacks and failures over the network. Then the normal behaviour would be no failure. If we observe the tests we realize that with these parameter configurations there are always some alarms executed.

First group of tests gives the best results for the tests 2 and 5. Probability of Alarm is lower in these cases. Furthermore, test 2 is better as test 5. As we can observe, the best parameters values during this group of tests are:

p	% of learning	*p_anom*	*T*	T_s	T_w
1	80	0.9	2	15	5

Table 18

In case of using more samples (as it was done in test 5), we can observe that the result is good too. But I decided to use window period of 5 minutes, and the sampling period of 15 seconds because these are the values proposed in the paper.

During the second group of tests, I applied flow normalization and attacks (marked in the figures with red circles). However the results are not the same than in previous case. In this case the best tests are those which better approach to the alarms executed. So I think they are more important than tests in first group.

We need to work with the parameters which better fix the alarms activated. So the tests which bring the best results are tests 14 and 15. And comparing both, I decided the best one is 15, because it presents fewer alarms as 14, and the time of detection[9] is short. The parameters proposed in this case are:

p	% of learning	*p_anom*	*T*	T_s	T_w
1	40	0.9	6	15	5

Table 19

With these two tests I decided that the percentage of learning samples stands around 40%. But I was not very sure about the value of T, so the next block of tests tries to specify the best percentage and the best value of T.

The order of p, never changes, and the parameters T_s and T_w are 15sec and 5min respectively. The probability of good behaviour at the beginning doesn't changes.

4.2.1.2 Best parameters decision. Step 2.

I will try to specify with this block which is the best percentage of learning samples, and the best value for the T parameter. In the last three tests I decreased the number of attacks because of the special parameters in use.

[9] Time needed by the algorithm for detecting an alarm.

4.2.1.2.1 Test 17

	p	% of learning	*p_anom*	*T*	T_s	T_w
Test 17	1	40	0.9	4	15	5

Table 20

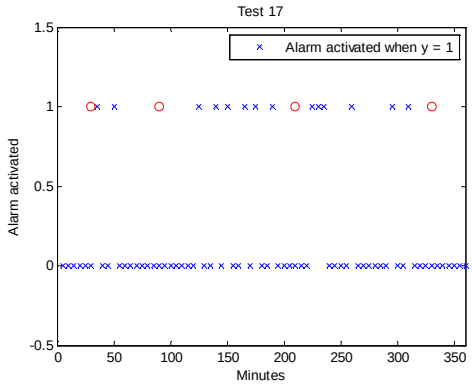


Figure 27

$$Probability\ of\ Alarm = 0.2206$$

$$Probability\ good\ behaviour = 0.7875$$

4.2.1.2.2 Test 18

	p	% of learning	*p_anom*	*T*	T_s	T_w
Test 18	1	30	0.9	3	15	5

Table 21

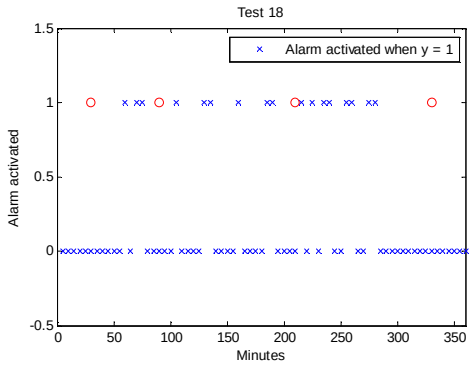


Figure 28

$$Probability\ of\ Alarm = 0.2609$$

$$Probability\ good\ behaviour = 0.7472$$

4.2.1.2.3 Test 19

	p	% of learning	*p_anom*	*T*	T_s	T_w
Test 19	1	30	0.9	1	15	5

Table 22

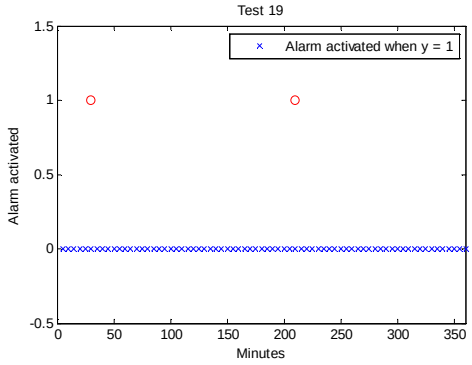


Figure 29

$$Probability\ of\ Alarm = 0$$

$$Probability\ good\ behaviour = 1$$

4.2.1.2.4 Test 20

	p	% of learning	*p_anom*	*T*	T_s	T_w
Test 20	1	30	0.9	2	15	5

Table 23

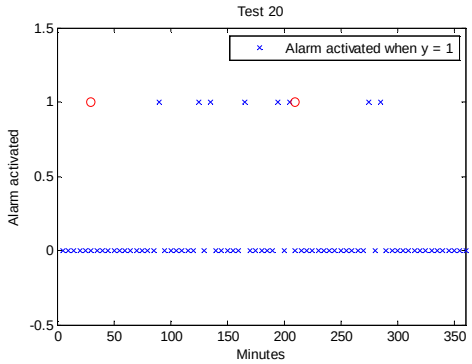


Figure 30

$$Probability\ of\ Alarm = 0.1304$$

$$Probability\ good\ behaviour = 0.8736$$

4.2.1.2.5 Test 21

	p	% of learning	*p_anom*	*T*	T_s	T_w
Test 21	1	30	0.9	8	15	5

Table 24

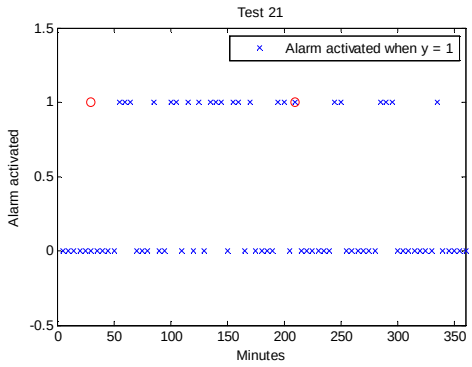


Figure 31

$$Probability\ of\ Alarm = 0.3478$$

$$Probability\ good\ behaviour = 0.6545$$

4.2.1.2.6 Test 22

	p	% of learning	*p_anom*	*T*	T_s	T_w
Test 22	1	30	0.9	5	15	10

Table 25

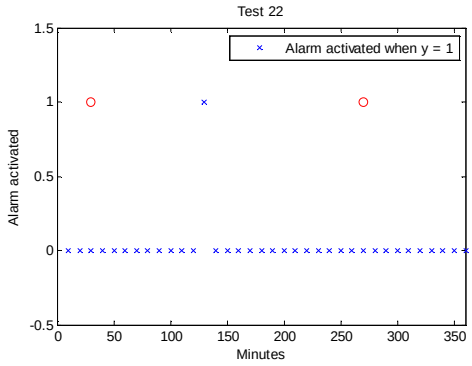


Figure 32

$$Probability\ of\ Alarm = 0.0313$$

$$Probability\ good\ behaviour = 0.96$$

4.2.1.2.7 Parameters conclusion

During the last block of tests we can decide that the best solutions are those given by tests 17 and 18. As we can see, not all the alarms are detected with these tests. However, they detect most of them, and the problem lies in the time of detection, because the tests finish before the time of detection has finished (The alarm which is never detected is the last one).

I decided test 17 gives better results as test 18 because the detection time is shorter in 17 than in 18. Therefore I established the value of percentage of learning samples and the value of memory of the system, T.

As I said, I would like to bring a guide for future developers and therefore I will present some rules or behaviour laws obtained from my experience. Of course they can be discussed with new results, which could bring a new perception of the problem.

During all the tests executions I could obtain some ideas. We can infer the next rules:

1) The memory of the system shouldn't be long. In this case the algorithm stabilises easily to the new situation avoiding false alarms. Therefore, as we could see, the tests with small T give better results in adaptability.
2) The best values for T_S and T_W are the values proposed in the paper. As we can see in the previous tests, when the window time decreases the number of false alarms increases, and in the opposite case, when the window time increases, not all the attacks are detected.
3) Probability of good behaviour at the beginning must be high when the algorithm starts. Observing tests 8 and 16, the audience can realize that the false alarms detected at the beginning are too much. Therefore, the less probability at the beginning, the more false alarms declared.
4) Decreasing the percentage of learning samples the behaviour also changes. We need to find the correct percentage which allows detecting almost all the alarms. It is obvious that the percentage is not 80% as it was at most of the tests run. In the third group of tests we decided that the best parameter is 40%.
5) In the case of the degree of the AR-Model, I decided not to change it because it was one of the design rules given by the writers in their paper.

Then, after testing the algorithm in these different situations, I am able to define which will be the best configuration parameters in order to obtain the best behaviour of the system.

The tests provided in next section are configured with the following best parameters:

p	% of learning	*p_anom*	*T*	T_s	T_w
1	40	0.9	4	15	5

Table 26

4.2.1.3 Anomaly situations with the best parameters

The results given in this section are execution of the algorithm with the best parameters. Here I only present the results obtained. The analysis of them is presented in section 4.2.2.

The paper presents some parameters which lend to define the algorithm behaviour. The table in the paper contains the following parameters

- Prediction Time:

  The writers of the paper, studies the frequency of different failures in their network. So that is why they present this parameter. In my case it has no sense, because the failures are produced by me, and I only consider the failures in those cases.

  I could have done what writers do, detect the anomaly using another process and compare the results with my own detection. But I supposed that the overload of work trying to synchronize both processes would be huge. Therefore I simplified the study.

- Detection time:

  This time is defined as the time that the algorithm needs for detecting failures. It is the time which starts when the anomaly happens and finish when the algorithm gives a false alarm.

- Time between false alarms:

  This is another control parameter which contains the time between two false alarms.

- Not detected alarms:

  I would like to add a new parameter in order to define the behaviour. This parameter extracts the number of alarms which could not be detected by the algorithm.

The parameters *detection time* and *time between false alarms* are presented in each of the following tests. They are the average of the times that they represent.

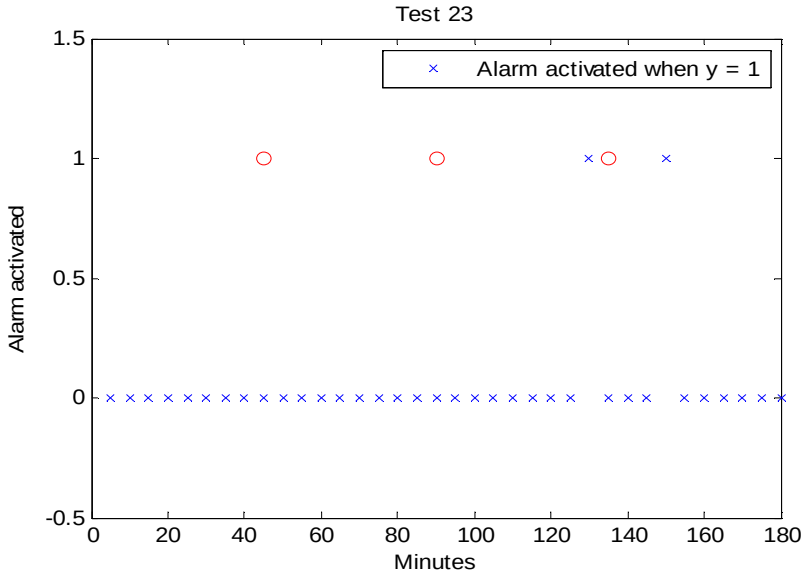
After the following test presentation, I will discuss about the results that writers obtain in the paper, and my own results. Later, I will discuss this algorithm comparing it with the other solutions proposed previously and ruled out because of the complexity.

4.2.1.3.1 Test 23

The alarms are activated in minutes of execution: 130 and 150.

The attacks were made in minutes of execution: 45, 90 and 135.

As we can observe, the successes happen in minutes 130 and 150. The rest are alarms not detected.



$$Average\ Detection\ Time = \frac{40\ \text{min} + 15min}{2} = 27'5min$$

$$Average\ time\ between\ false\ alarms = \ No\ false\ alarms\ detected$$

$$Not\ detected\ alarms = \ 1$$

$$Probability\ of\ Alarm = \ 0.0625$$

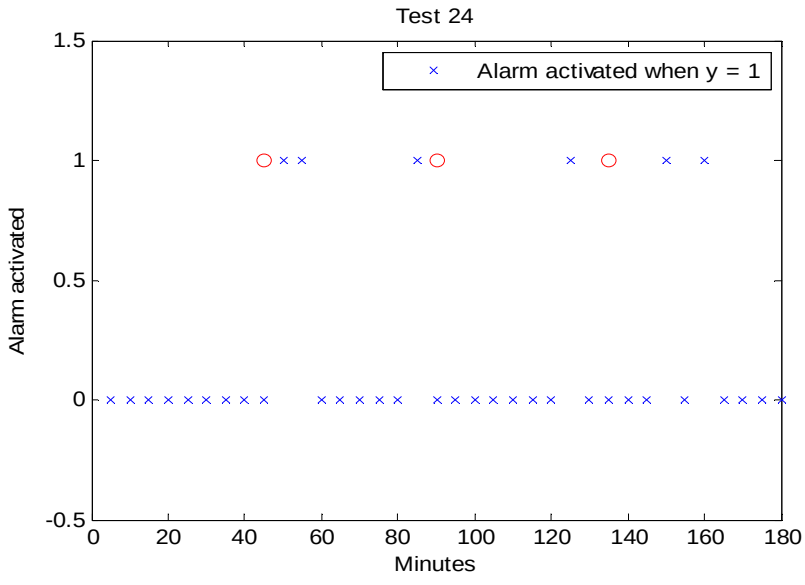
$$Probability\ good\ behaviour = \ 0.9343$$

4.2.1.3.2 Test 24

The alarms are activated in minutes of execution: 50, 55, 85, 125, 150 and 160.

The attacks were made in minutes of execution: 45, 90 and 135.

As we can observe, the alarms success are those in minutes 50, 125 and 150. The rest of them are false alarms.



$$Average\ Detection\ Time = \frac{5\ \text{min} + 35min + 15min}{3} = 18'3min$$

$$Average\ time\ between\ false\ alarms = \frac{25min + 75min}{2} = 50min$$

$$Not\ detected\ alarms = 0$$

$$Probability\ of\ Alarm = 0.1875$$

$$Probability\ good\ behaviour = 0.8250$$

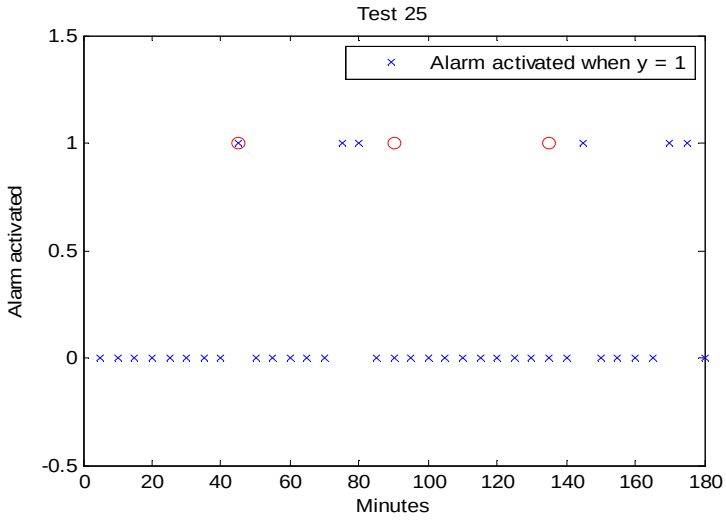
The results in this case bring fewer alarms activated than in other cases, but we are not sure if the algorithm has detected all the failures. Let's continue with the rest of the tests.

4.2.1.3.3 Test 25:

The alarms are activated in minutes of execution: 45, 75, 80, 145, 170 and 175.

The attacks were made in minutes of execution: 45, 90 and 135.

As we can observe, the alarms success are those in minutes 45 and 145. The rest of them are false alarms. There is an attack not detected



$$Average\ Detection\ Time = \frac{0\ \text{min} + 10min}{2} = 5min$$

$$Average\ time\ between\ false\ alarms = \frac{5min + 90min + 5min}{3} = 33'3min$$

$$Not\ detected\ alarms = 1$$

$$Probability\ of\ Alarm = 0.1875$$

$$Probability\ good\ behaviour = 0.8250$$

4.2.2 Analysis of results and comparison with other solutions

As we could observe in the results are not as precise as we thought. In this section I am comparing the results of my work with the result given by the paper. I will present the efficiency given by other solution proposed above.

If we join the results obtained by the tests 23, 24 and 25 the values of the parameters are:

Average detection time	Average time between false alarms	Not detected alarms/Number of alarms	Probability of alarm	Probability of good behaviour
16.93 min	41.65min	2/9	0.1458	0.8614

Table 27

Now the results proposed in the algorithm joining all the possible failures in the system:

Anomaly Type	Avg.Prediction Time T_p(mins)	Avg.Detection Time T_{dt}(mins)	Avg.Mean Time Between False Alarms T_f(mins)
File server failures	26	30	455
Network Access problems	26	23	260
Protocol Error	15	-	-
Runaway process	1	-	235

Figure 33

Average detection time	Average time between false alarms
26.5min	316.6min

Table 28

We can observe that the results are not very similar. This is probably caused by several factors, which we will see below. The detection time I obtained is around the order of the result presented in the paper. For my parameter definition this time is shorter, but on the other hand the time between false alarms differs too much.

The detection time is very important for the algorithm because if there are two failures very close, we will probably not detect the second failure. This is one of the characteristics which makes this algorithm weak.

I must explain that in my case I only try to detect one type of failures because of the limited environment where I run my tests. It is possible that some of the alarms which I decided as

false alarms are, in fact, real alarms caused by other kinds of failures. It is necessary to realize that the writers do not use the attacks to the network in order to obtain their results.

Other explanation is a bad decision over the best parameters calculated. I needed to make too many tests to obtain the best configuration parameters, and I made several suppositions which could be wrong. These suppositions were not solved by the writers of the paper when I asked for them. The future developers could continue with my work and try to accuracy the functionality of the algorithm.

I am talking now about the adaptability of the algorithm. Not all the attacks are detected, but most of them are detected independently on the time were they occurs. That means that the algorithm adapts to the normal behaviour of the network. When the network changes the behaviour and it receives an attack, the algorithm does not consider the change in the behaviour but the attack. This can be observed in the tests, because the alarms are detected in most of the cases in a short period of time.

I will focus now in the other kinds of algorithms and the features which made them more suitable or not to be implemented instead of the method proposed in this work.

- Maximum entropy method:

  This method is very precise as the writers say. The precision in their experiments stands around 0.9 (where 1 is the highest value).

  It brings better results than the one chose for this work, but more overload to the system which manages it.

  The problem of this algorithm is the difficult understanding it. *The method requires a constant memory and a computation time proportional to the traffic rate. Many interesting aspects of this approach still remain to be explored, and comparison with other methods such as Holt-Winter, when possible, will be useful.* (6)

- Network anomalies using sketch subspaces

  In this case, the probability of non-detection of anomalies with a network of about 20 or less computers stands around 0.2.

  This method can detect IP flows and specify which flow is causing the failure. The time needed for detecting an anomaly is short.

  The Defeat algorithm presented in this paper uses multiple random traffic projections to robustly detect anomalies. In a week-long trace from the Dante backbone, Defeat detects nearly 200 more anomalies than prior work while missing only one. It is, in addition, able to automatically infer the IP flows responsible for an anomaly, a feature missing in any previously published work. Defeat brings on-line anomaly detection and identification within the realm of feasibility, which forms our future work. (7)

- ARP-based Anomaly Detection Algorithm Using Hidden Markov Model:

  The writers propose three different methods using this technique. They present the results in function of what they call false alarm rate. This rate is the probability of false alarm which can be introduced in the system. The variation of this rate produces a variation of the anomaly detection rate. The highest value of this rate, the best behaviour of the algorithm in the network.

  For the third method the anomaly detection rate stands around 0.98. This is too much better than the algorithm proposed.

5 Conclusion and Further Developments

The main objective of this work was to find an adaptive algorithm able to change the detection threshold as the network changes. I found the solution with the algorithm proposed and developed. This algorithm adapts its behaviour to new situations in the network and detects all kind of failures produced in a system.

The accuracy of this method is not as good as in other solutions, but it is easily developed in almost any kind of IP network. The configuration of the algorithm is a hard step to carry out, but when the algorithm is running it is very stable.

This algorithm was chose because I found it simple to understand at the beginning. I read the other algorithms briefly as well, and finally I decided to implement this one, because the idea was clear in my mind. Later I found some complex points which I didn't realize. But at the end the solution was possible and helpful for the understanding of this kind of algorithms. It is a good choice with a network with not too many requirements.

The algorithm can detect broadly speaking the place where the anomaly happens. In fact the algorithm bases itself in an analysis of each network element, and detects the anomaly on it. It means that the failure happens in the area near the network element. Actually the algorithm detects the area of influence of the failure. This feature cannot be found in this work, because I only worked with one network element, but it can be easily solved by adding a manager which reads all the alarms and give the places of failure to the human manager.

Future works: Going deeply in this area is possible, but the new adapting algorithms work with other signal processing techniques, which seem to be more powerful and efficient. However the correct development of this work can be helpful and simple. I hope this work could help new developers to design this kind of systems faster and easier.

It might be possible to design this algorithm in non-IP networks, if the data acquirement changes and the information is obtained from other traffic.

5.1 Viability

The work carried out has been tested in a small environment, as it was explained. But the solution obtained can be extrapolated to a big network. The same algorithm can manage some network elements, but it has a top limit of them which depends on the parameters used to configure the algorithm.

If the network is higher than the limit established by the algorithm, then the network can be divided in areas, creating an agent (machine which runs the algorithm) in each area. In this case there must be a general machine which receives the alarms detected by the different agents and processes the information given to the human manager.

The system might be chaotic in a big environment if the algorithm is not well configured. The number of false alarms detected can confuse the manager, and bother so much. It is very important the good configuration of the algorithm and a specific planning for the network.

This system can be implemented in many IP networks, because the protocol used for the data acquisition can be found in almost all network elements. It is very important to find out if our network works with SNMP before developing the system.

This algorithm has been tested only over one router. If we wanted to run it over other network elements, we should find out which are the best MIB variables for each kind of network element. The writers propose to collect data from the element and run a Principle component analysis in order to specify which the best variables are.

The future developers should not take this work as the only reference. There are too many bibliographies available about this field.

APPENDIX: Spanish summary

A. Introducción

Este proyecto ha sido desarrollado en el departamento RUS de la Universidad de Stuttgart, bajo la supervisión de Antonio Cuevas y Patrick Mandic. La duración del mismo ha abarcado 7 meses desde el comienzo hasta la última presentación. Durante su desarrollo, se han realizado dos exposiciones, una a los cuatro meses presentando la solución propuesta, y otra al final haciendo un resumen general del proyecto, implementación y resultados.

El departamento RUS, se encarga de monitorizar las redes de la universidad y a su vez realiza proyectos de comunicaciones telemáticas con empresas y universidades. Participan en el proyecto Daidalos (**D**esigning **A**dvanced network **I**nterfaces for the **D**elivery and **A**dministration of **L**ocation independent, **O**ptimized personal **S**ervices) para el cuál poseen una red de ordenadores dedicada.

Es en esta red donde se me propone el objetivo de mi proyecto. Diseñar un sistema capaz de detectar anomalías en la red de manera adaptativa. Para ello tuve que realizar un primer trabajo de investigación sobre las tecnologías existentes en este campo y posibles desarrollos. En segundo lugar se hizo necesaria la elección de un algoritmo que permitiera solucionar el problema de manera eficiente, e implementarlo en la red propuesta. Por último el problema se centró en la experimentación con este sistema y la comparación de la solución con el resultado esperado.

Vamos a aclarar por el momento que era realmente un sistema adaptativo de detección de anomalías en la red.

Detección de anomalías en red: También conocida como análisis de comportamiento de la red, la detección de anomalías está diseñada para proporcionar una clara visión del comportamiento de la red y detectar automáticamente las amenazas activas a la seguridad, el comportamiento de usuario arriesgado, las cuestiones de rendimiento y las actividades que no siguen las normas, como el acceso a áreas restringidas y los cambios en la red no autorizados. Estos sistemas examinan continuamente el comportamiento de los usuarios, la red y las aplicaciones.

Existen dos tipos de sistemas para detectar anomalías: adaptativos y no adaptativos.

- **Sistemas no adaptativos:** Este tipo de sistemas son los primeros que se propusieron para solucionar este tipo de fallos en red. Se basan en la definición de reglas para la identificación de los distintos ataques y fallos. Cuando un nuevo fallo o ataque es identificado, el administrador de la red debe definir cómo se comporta este nuevo fallo o ataque e inducir un grupo de reglas que permitan identificarlo en un futuro. Cuando estas reglas son introducidas en un sistema no adaptativo, el sistema será capaz de decidir si el tráfico se comporta de forma normal o si está sufriendo este nuevo fallo. Así con todo, este tipo de sistemas necesitan muchas actualizaciones a lo

largo de su vida para poder seguir funcionando correctamente, y el administrador de la red precisa de mucho tiempo para configurar el sistema.

- **Sistemas adaptativos:** Estos sistemas aparecieron algo más tarde que los no adaptativos intentando introducir tratamiento de señal sobre la información obtenida de la red con el fin de identificar un comportamiento normal de la red y actuar sobre los comportamientos anómalos (fallos en la red). La ventaja de este tipo de sistemas radica en que no hace falta que sean actualizados, ya que van adecuándose al nuevo comportamiento de la red según pasa el tiempo, y cada cambio sobre este comportamiento se considera un fallo.

Como ejemplo de sistemas no adaptativos estudié un sistema que funciona actualmente en la red de la Universidad de Stuttgart, Arbor Peakflow. Es un sistema de detección avanzado que permite al administrador de red descargarse las reglas que definen los fallos más comunes y ataques detectados últimamente en redes. Pese a todo, el administrador necesitará definir sus propias reglas para cierto tipo de comportamientos en su red.

También realicé un estudio sobre los sistemas adaptativos, tal y como los definiré a continuación:

- **Método de la máxima entropía:** Este algoritmo se obtuvo del trabajo de Yu Gu, Andrew McCallum y Don Towsley (6). Comparan la información obtenida de la red con una distribución *baseline*. Esta comparación se realiza con el método de la máxima entropía. Con esta tecnología se obtiene una nueva ventaja ya que el algoritmo se da cuenta de cuando existe un cambio lento o abrupto en el comportamiento de la red. Esto implica que el tipo de anomalía puede ser detectada.
- **Detección de anomalías en la red usando aproximación por subespacios:** Xin Li, Fang Bian, Mark Crovella, Christophe Diot, Ramesh Govindan, Gianluca Lannaccone y Anukool Lakhina proponen la técnica en (7). La realización de este algoritmo se basa en dividir el tráfico en trozos que son aproximados por el método de los subespacios. Los autores han presentado unos resultados donde puede apreciarse que la precisión del algoritmo es muy alta. Además esta técnica es capaz de detector que flujos causan la anomalía y cuál es la IP de origen de la anomalía.

- **Algoritmo de detección de anomalías basado en tráfico ARP usando el Modelo oculto de Markov:** Este método utiliza la información proporcionada por el tráfico ARP para identificar la anomalía. El algoritmo de tratamiento de señal propuesto para este caso es el Modelo oculto de Markov. El desarrollo de este algoritmo se puede encontrar en el *paper* "An ARP-based Anomaly Detection Algorithm Using Hidden Markov Model in Enterprise Networks" (8).

- **Detección de anomalías en redes IP usando modelos AR:** Esta técnica basa su idea en la detección de cambios abruptos sobre variables MIB predefinidas, que representan el estado en el que se encuentra el elemento de red sobre el que se aplica el algoritmo. Puede encontrarse más información en el *paper* "Anomaly

detection in IP networks" (1). Por supuesto también en el presente trabajo, ya que este es el algoritmo seleccionado para su desarrollo.

B. Descripción del algoritmo e implementación.

El algoritmo propuesto en el último apartado de la introducción fue elegido porque se consideró que era el más sencillo de implementar y entender. El *paper* era fácil de entender y me decidió la posibilidad de escalar el sistema al tamaño de nuestra red. Decidí implementarlo y probar como se comportaba en el entorno que yo disponía en la Universidad de Stuttgart.

Como se define en el trabajo el entorno de desarrollo de este algoritmo sería una cosa como la siguiente:

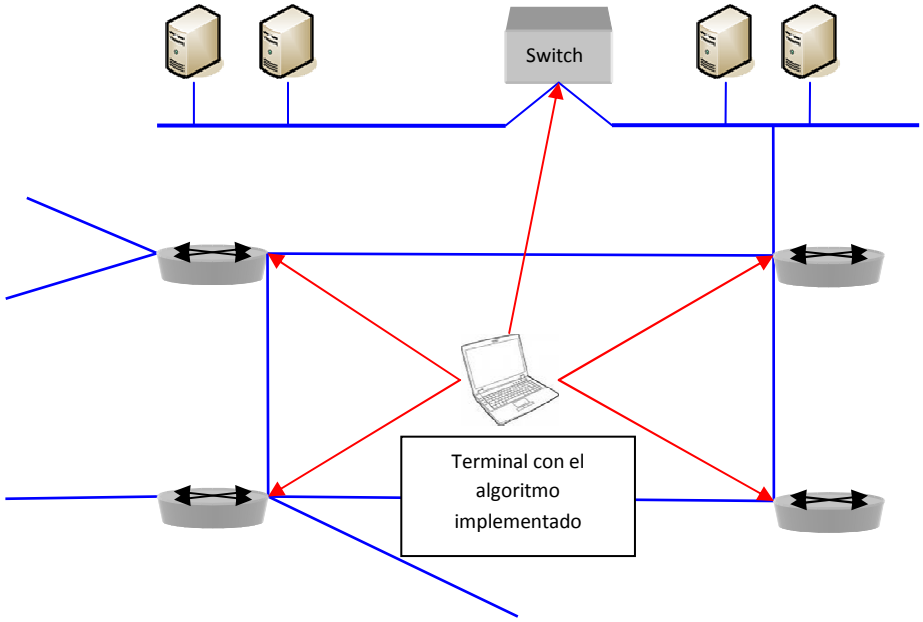


Figure 34

Como se puede observar el sistema realiza el algoritmo a cada uno de los elementos de red que conforman la red que queremos monitorizar. Estos elementos de red son principalmente routers y switches. Para cada uno de estos elementos decidimos si su comportamiento es normal o anormal y podemos definir el lugar aproximado donde se produce la anomalía.

El algoritmo se basa en el siguiente esquema de funcionamiento. El programa obtiene los valores de las variables MIB predefinidas cada tiempo de muestreo, durante un tiempo de ventana mayor que este tiempo de muestreo. La información es almacenada en un fichero de texto para ser posteriormente procesada.

Cada tiempo de ventana el algoritmo obtiene un valor que indica si hay una anomalía en la red o no. Estos valores (tiempo de ventana y tiempo de muestreo) son, entre otros, parámetros de diseño.

Lo que hace el algoritmo durante un tiempo de ventana para obtener este valor de alarma o no se define a continuación. Una vez que los datos de una ventana han sido almacenados se procesan aplicando el algoritmo.

Lo primero que habría que hacer para detectar una anomalía es identificar cambios abruptos en la variable MIB, esto significa que cambia drásticamente el valor que va teniendo normalmente. Para detectar estos cambios abruptos usamos la teoría de modelos AR (modelos autorregresivos). Consideraremos la serie de valores que toma la variable durante el tiempo de ventana y supondremos que es un modelo AR. Obtenemos la varianza obtenida al considerar esta serie un modelo autorregresivo.

Por otro lado consideramos la misma serie de datos como dos series, una de aprendizaje que será un número de muestras del comienzo (parámetro de diseño del algoritmo), y otra que es de test compuesta por el resto de muestras hasta el fin de ventana. Para estas dos series procederemos de igual forma que en el caso anterior. Se consideran modelos AR y se obtienen sus varianzas asociadas.

Mediante el método GLR (Generalized likelihood ratio) aplicado sobre las tres varianzas obtenidas conseguiremos un parámetro conocido como el indicador de abnormalidad asociado a la variable MIB correspondiente.

Realizando este proceso sobre las N variables MIB que tengamos para el elemento de red, vamos a obtener un vector compuesto por los indicadores de abnormalidad obtenidos de cada variable y un indicador de probabilidad de buen comportamiento de la red, con una dimensión de (N+1)x1.

El siguiente proceso implica la acumulación de los últimos vectores obtenidos en pasadas ventanas y el obtenido en la ventana actual. El número de vectores pasados que se utilizan es también un parámetro de diseño.

Una vez seleccionado el número de vectores se procede al cálculo de la matriz A a partir de estos vectores. Esta matriz es fundamental ya que mediante sus autovectores puede definirse la región de fallo y si el elemento está en esta región de fallo. Si lo está, quiere decir que se ha detectado una anomalía en el elemento durante la ventana que nos ocupa.

La implementación se ha realizado sobre Matlab, la parte de cálculo y sobre Shell-script la parte de adquisición de datos. El sistema se ha implementado sobre Linux debido a que ofrece muchas soluciones de software libre.

Matlab ha sido escogido por poseer implementadas las funciones de cálculo necesarias. Se pierde cierta eficiencia al elegir este lenguaje, pero se gana en simplicidad de código. Hay funciones que implementadas en c reducirían también bastante la eficiencia.

En la siguiente figura puede observarse un esquema de los procesos que se llevan a cabo para comunicar Matlab con Shell-script.

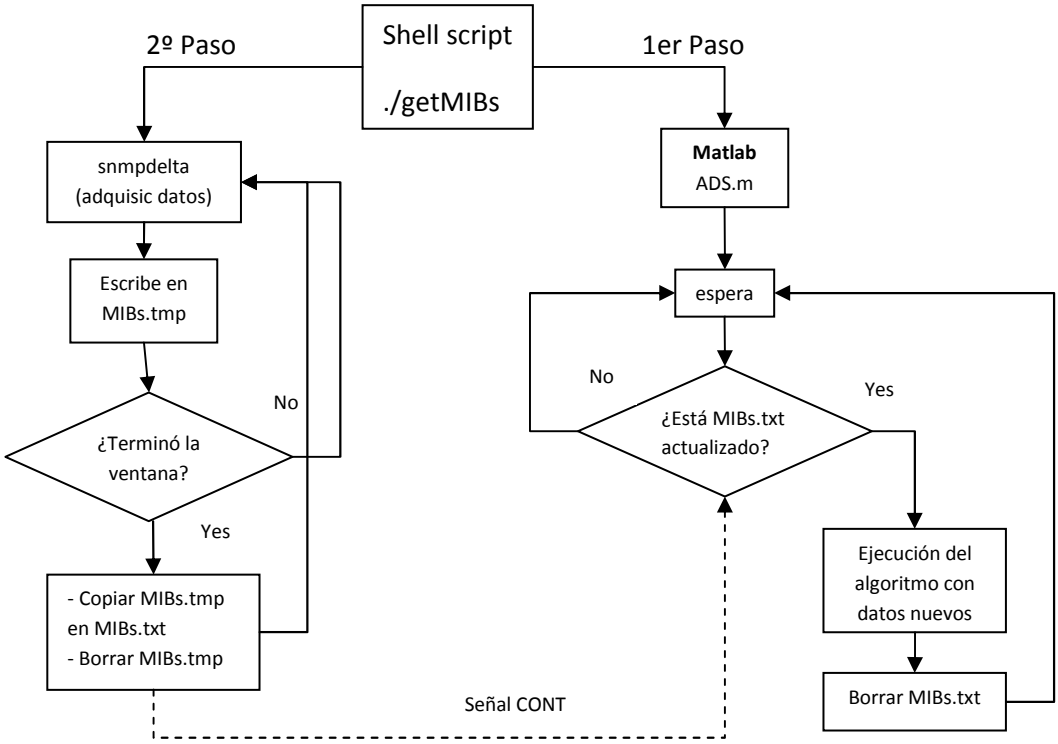


Figure 35

C. Desarrollo Práctico.

En mi caso este entorno era teórico ya que el entorno disponible para mis pruebas era mucho más simple. Esto pudo afectar en cierto modo a los resultados que presentó el sistema pese a que parecía que la escalabilidad del sistema era buena.

El entorno disponible en la universidad estaba compuesto un sólo router por el cual pasaba todo el tráfico de la red. Mi desarrollo sólo consideraba este router como nodo de red sobre el que aplicar el algoritmo. Así que el comportamiento era bastante limitado. Entonces fue cuando decidí enfocar el trabajo como una guía de desarrollo para futuras implementaciones, siguiendo con la consideración de que la escalabilidad era buena.

Para el desarrollo práctico decidí explorar otra solución para la toma de datos. Es decir obtener la información a través de otro sitio que no fuera el protocolo SNMP. Así que investigué la obtención de la información a través de los flujos que atravesaban el router. Esta solución hacía mucho más complejo el desarrollo del algoritmo además de ineficiente.

Después de haber desarrollado el software debemos comprobar que funciona correctamente por lo que el entorno de pruebas se divide en dos fases. Una primera en la que se intentan aproximar los parámetros de diseño a su mejor valor para que el funcionamiento en nuestro entorno de pruebas sea lo más correcto posible. La segunda

consiste en el test del sistema una vez que este ha sido configurado con los mejores parámetros de diseño hallados.

Esto se debe a que los autores no proponen valores para estos parámetros de diseño, de modo que debemos investigarlos nosotros, aunque propone varios entornos de aplicación y distintas pruebas con varios tipos de fallos de red.

Como he comentado previamente, el desarrollo del banco de pruebas se llevó a cabo en el departamento RUS de la Universidad de Stuttgart, en la red habilitada para el proyecto Daidalos. Para mi trabajo recibí un portátil sobre el que tuve que instalar las aplicaciones necesarias y especificadas antes además de una conexión al router para probar y experimentar tanto como quisiera.

Por lo tanto el entorno de trabajo viene presentado en la siguiente figura:

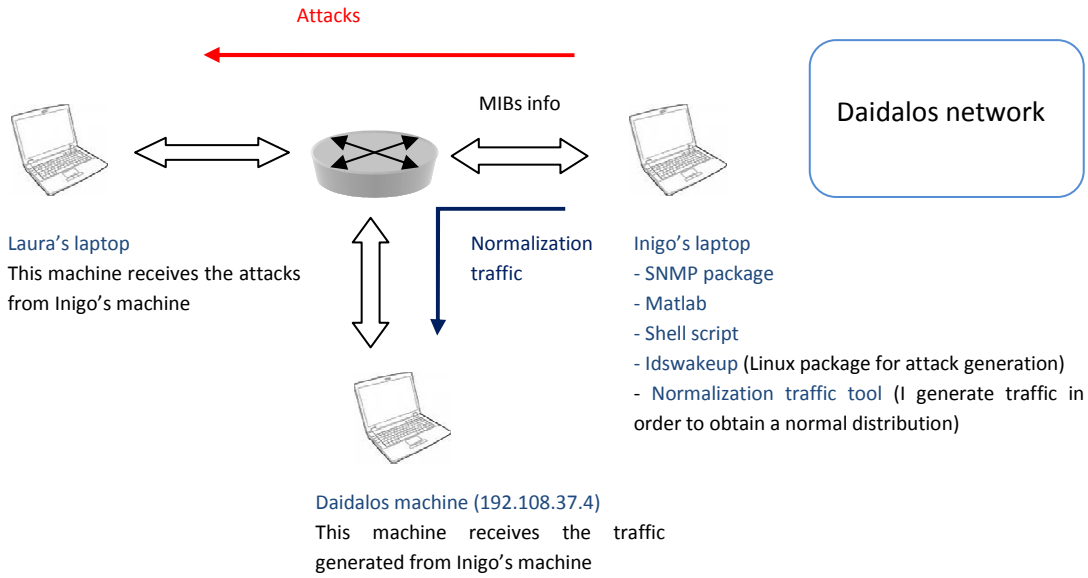


Figure 36

A la hora de simular los fallos en la red el entorno de trabajo del que disponía se quedaba demasiado corto. No podía desconectar ningún servidor para ver como se comportaba el sistema ya que estos nodos de la red realizaban tareas importantes. Los únicos fallos que podía simular y controlar en la red fueron ataques que realizaba desde mi mismo equipo a otro equipo. Estos ataques eran inofensivos aunque producían un comportamiento anormal en la red, con lo que era capaz de detectar y afinar la precisión de mis parámetros de diseño.

La red utilizada no tenía un comportamiento demasiado regular, por lo que mis tutores me propusieron que generara un tráfico de normalización para trabajar sobre una red de comportamiento más uniforme.

Como entorno de pruebas, realicé un grupo de test para obtener el valor de los parámetros de diseño que obtienen los mejores resultados. Esta parte se considera una de las más importantes del diseño, ya que se deduce de la misma cómo afectan los parámetros sobre el resultado del algoritmo.

Esta última parte se considera muy importante en la misión para la que se ha realizado el proyecto. Ofrece unas guías de comportamiento para los parámetros y el algoritmo de modo que futuros desarrolladores obtengan los mejores resultados sólo con seguir estas directrices, evitándoles muchas horas de configuración y test.

Se realizan tres grupos de test para la configuración del sistema. Los test realizados en los dos primeros grupos tienen una duración de 12 horas. En el tercer grupo cada test se ejecuta durante 6 horas.

Los dos primeros grupos de test ejecutados en la configuración de parámetros pueden verse en la siguiente tabla.

	AR-Model order (p)	Percentage of learning samples $\left(\frac{N_L}{N_L+N_S}\right)$	Initial probability of good behaviour (p_anom)	Memory of the algorithm (T)	Sampling period in seconds (T_s)	Window time in minutes (T_w)
Tests 1 & 9	1	80	0.9	6	15	5
Tests 2 & 10	1	80	0.9	2	15	5
Tests 3 & 11	1	80	0.9	15	15	5
Tests 4 & 12	1	80	0.9	6	3	2
Tests 5 & 13	1	80	0.9	6	45	20
Tests 6 & 14	1	20	0.9	6	15	5
Tests 7 & 15	1	40	0.9	6	15	5
Tests 8 & 16	1	80	0.5	6	15	5

Table 29

Los primeros ocho test se ejecutan sobre la red directamente sin considerar ataques como fallos, es decir, intentan identificar simplemente mal comportamiento en la red sin saber con certeza si realmente se ha dado este mal comportamiento. Se considera que hay un buen comportamiento del algoritmo en este grupo cuando la probabilidad de alarma en la red es lo más baja posible. La consideración es que no existen fallos en la red durante el tiempo de test.

Los siguientes ocho se han realizado con normalización de tráfico sobre el elemento de red y con ataques de una interfaz del mismo a otra. En este caso sabemos cuándo se realizan los ataques, por lo tanto, el buen comportamiento es cuando las alarmas se activan en poco tiempo a la ocurrencia de un ataque.

Esta tabla ofrece una primera aproximación de parámetros.

p	% of learning	p_anom	T	T_s	T_w
1	40	0.9	6	15	5

Table 30

A partir de ellos se ejecuta un nuevo grupo de test que tratan de afinar más el comportamiento del algoritmo. A continuación se referencia la configuración de parámetros probada para cada test.

	AR-Model order (*p*)	Percentage of learning samples $\left(\frac{N_L}{N_L+N_S}\right)$	Initial probability of good behaviour (*p_anom*)	Memory of the algorithm (*T*)	Sampling period in seconds (T_s)	Window time in minutes (T_w)
Tests 17	1	40	0.9	4	15	5
Tests 18	1	30	0.9	3	15	5
Tests 19	1	30	0.9	1	15	5
Tests 20	1	30	0.9	2	15	5
Tests 21	1	30	0.9	8	15	5
Tests 22	1	30	0.9	5	15	10

Table 31

Aparte de estos test, se han obtenido los resultados que daba el sistema para los mejores parámetros que se ha deducido que son los siguientes.

p	% of learning	*p_anom*	*T*	T_s	T_w
1	40	0.9	4	15	5

Table 32

Esta configuración aproxima los mejores resultados del algoritmo sobre mi sistema. En el desarrollo del trabajo se han facilitado unas directrices del comportamiento de estos parámetros en sistemas de modo que sea fácil su configuración.

Una vez obtenidos estos parámetros se ejecutan tres test de tres horas cada uno intentando identificar los ataques. A partir de estos resultados, comparamos con los resultados esperados por el *paper*.

Los resultados obtenidos en mi caso han sido:

Tiempo de detección medio	Tiempo medio entre falsas alarmas	Alarmas no detectadas/Total alarmas	Probabilidad de alarma	Probabilidad de buen comportamiento de la red
16.93 min	41.65min	2/9	0.1458	0.8614

Table 33

Estos resultados deben ser comparados con los esperados en el trabajo en el que me he basado.

Tiempo de detección medio	Tiempo medio entre falsas alarmas
26.5min	316.6min

Table 34

Como puede observarse el tiempo de detección medio mejora en mi trabajo pero a su vez mi algoritmo detecta más falsas alarmas debido a que el tiempo medio entre falsas alarmas es mucho más pequeño.

Comparando estos resultados con los obtenidos mediante otras técnicas más complejas llego a la conclusión de que el algoritmo propuesto se puede implementar pero los resultados no son demasiado buenos. Otros algoritmos como los propuestos también en este trabajo permiten llegar a mayores precisiones.

Otra conclusión obtenida es que al incrementar levemente la complejidad del algoritmo seleccionado, la eficiencia obtenida mejora en mayor medida. Para futuros proyectos de desarrollo en este entorno, propuse al departamento del RUS en Stuttgart implementar otros algoritmos que mejorarían mucho los resultados.

D. Conclusiones

El objetivo principal de este trabajo era encontrar un algoritmo adaptativo que fuera capaz de cambiar los umbrales de detección de fallos en una red a lo largo del tiempo. La solución encontrada ha sido propuesta e implementada como trabajo. A su vez se ha estudiado su comportamiento para ver si funcionaba igual que las expectativas teóricas. Como ha podido observarse el algoritmo es capaz de detector fallos y situaciones extrañas en la red y adaptarse a las mismas.

Si el sistema empieza a detectar muchas veces el mismo fallo o comportamiento anómalo, lo más probable es que acabe considerándolo un comportamiento normal. Véase el caso en que se introduce una nueva aplicación en la red que genera un tráfico sospechoso. El administrador de la red verá que el algoritmo genera una alarma pero sabiendo que acaba de instalarse una nueva aplicación, lo más probable es que no necesite hacer caso al algoritmo y esperará a que este se estabilice con la nueva situación.

Como trabajo futuro se puede seguir desarrollando este sistema para obtener mejores resultados o desarrollar nuevas técnicas de procesamiento de señal que ofrezcan mejores resultados. Sin embargo el correcto desarrollo de este sistema propuesto en un entorno más realista puede fácilmente solventar el problema propuesto y se ofrece como un sistema con poca carga de trabajo y efectivo. Aunque para redes que requieran una mayor precisión y aproximación se hará necesario otro tipo de algoritmos más complejos.

Espero que con mi estudio se facilite el trabajo de futuros desarrolladores y sirva como una guía que permita reducir el tiempo de estudio de este tipo de sistemas. Como digo no es un trabajo totalmente finalizado ya que aún queda mucho por desarrollar en este campo. Estudios como este, espero que ayuden a clarificar un poco el problema que se presenta y den una idea simple de los algoritmos y procedimientos existentes.

E. Contacto y agradecimientos

Me gustaría agradecer el apoyo recibido por mi familia y amigos. También a todos aquellos que directa o indirectamente participaron en el desarrollo de mi trabajo y me ayudaron a su mejor comprensión, como el Departamento RUS en Stuttgart, y en especial a mis tutores Antonio Cuevas y Patrick Mandic.

Un agradecimiento también a mi compañera Laura Herranz que realizó un proyecto parecido al mío durante el mismo tiempo en la universidad de Stuttgart, por ayudarme en la comprensión y el desarrollo del trabajo.

También me gustaría agradecer su ayuda y seguimiento a mi tutor en la universidad Carlos III, Jose Ignacio Moreno. Y a los servicios de esta universidad al resolverme dudas académicas.

Agradezco también la contestación a mis dudas que recibí de una de las autoras del trabajo en el que basé mi proyecto, Marina Thottan.

Para mayor referencia se me puede contactar por la dirección de email:

inigo.ucero@gmail.com

Sin más preámbulo finalizo aquí el desarrollo de mi Proyecto de Fin de Carrera desarrollado en la universidad de Stuttgart.

F. Información de interés.

El trabajo ha sido realizado en la universidad de Stuttgart por Iñigo Ucero Aristu, estudiante de Ingeniería de Telecomunicación en la universidad Carlos III de Madrid. Los coordinadores académicos en la universidad de Stuttgart son Antonio Cuevas y Patrick Mandic. El cotutor en la universidad Carlos III es Jose Ignacio Moreno Novella.

La presentación del trabajo y lectura del mismo se realizó el dia 29 de Septiembre del año 2008. La calificación obtenida en escala alemana ha sido 1’3.

Bibliography

1. **Marina Thottan and Chuanyi Ji.** *Anomaly detection in IP networks.* s.l. : IEEE Trans. Signal Processing 51, 2003.

2. **Matthew Tanase, President of Qaddisin.** The State of anomaly detection. [Online] http://www.securityfocus.com/infocus/1600.

3. Arbor Specifications (“a1_main_ts.pdf”, “a2_initial.pdf”, “a3_architecture.pdf”). s.l. : Arbor Docs.

4. Arbor Users Guide (“u1_technov.pdf”, “u2_gui.pdf”, “u3_rule.pdf”, “u4_network_vis.pdf”, “u5_workflows.pdf”, “u6_reports”, “u7_admin_set”). s.l. : Arbor Docs.

5. Arbor Web Page. [Online] http://www.arbornetworks.com/peakflowsp.

6. **Yu Gu, Andrew McCallum and Don Towsley.** *Detecting Anomalies in Network Traffic Using Maximum Entropy Estimation.* s.l. : Department of Computer Science, University of Massachusetts, Amherst, MA 01003.

7. **Xin Li, Fang Bian, Mark Crovella, Christophe Diot, Ramesh Govindan, Gianluca Lannaccone and Anukool Lakhina.** *Detection and Identification of Network Anomalies Using Sketch Subspaces.* Rio de Janeiro, Brazil : In ACM Internet Measurement Conference, October 2006.

8. **Dat Tran, Wanli Ma, and Dharmendra Sharma.** *An ARP-based Anomaly Detection Algorithm Using Hidden Markov Model in Enterprise Networks.* s.l. : IJCSNS International Journal of Computer Science and Network Security, VOL.8 No.2, February 2008.

9. Download UBUNTU. [Online] http://www.ubuntu.com/getubuntu/download.

10. **M. Rose and K. McCloghrie.** *RFC 1155: Structure and Identification of Management Information.* May 1990.

11. **D. Harrington, R. Presuhn and B. Wijnen.** *RFC 3411: An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks.* December 2002.

12. **R. Presuhn, J. Case,K. McCloghrie, M. Rose and S. Waldbusser.** *RFC 3418: Management Information Base (MIB) for the Simple Network Management Protocol (SNMP).* December 2002.

13. Cisco NetFlow Datasheet. [Online] http://www.cisco.com/en/US/prod/collateral/iosswrel/ps6537/ps6555/ps6601/product_data_sheet0900aecd80173f71.html.

14. How to… FlowTools. [Online] http://www.dynamicnetworks.us/netflow/documents/netflow-howto-v1.4a.pdf.

15. FlowTools manual. [Online] http://www.splintered.net/sw/flow-tools/docs/.

16. **M. Basseville and I. Nikiforov.** Detection of Abrupt Changes, Theory and Application. s.l. : Englewood Cliffs: Prentice-Hall, 1993.

17. FP6 IST Integrated Project Daidalos. [Online] http://www.ist-daidalos.org/.

18. **A. S . Williky and H. L. Jones.** *A generalized likelihood ratio approach to the detection and estimation of jumps in linear systems.* s.l. : IEEE Trans. Automat. Contr., pp. 108-112, 1976.

19. **Gustafsson, Fredrik.** *The Marginalize Likelihood Ratio test for Detecting Abrupt Changes.* JANUARY 1996 : IEEE TRANSACTIONS ON AUTOMATIC CONTROL, VOL. 41, NO. 1.

20. **Marina Thottan and Chuanyi Ji.** *Adaptive Thresholding for Proactive Network Problem Detection.* s.l. : In Proceedings of the IEEE Third International Workshop on Systems Management, page 108. IEEE Computer Society, 1998.

Acknowledgment

The author would like to thank the help received by all the people working in the department of RUS in the University of Stuttgart. Specially, I would like to thank the work of my supervisors Antonio Cuevas and Patrick Mandic. I apologize for any inconvenience I could have done to anybody in the department during my period of work.

Thanks to Laura Herranz, my partner, for the help and the encourage received and the work together. The ideas of her work could help me to end properly mine.

Finally, I would like to thank the writers of “Anomaly detection in IP-networks”, Marina Thottan and Chuanyi Ji, the help received during the development of my work.

Declaration

Herewith, I declare that I have developed and written the enclosed thesis entirely by myself and that I have not used sources or means except those declared. This thesis has not been submitted to any other authority to achieve an academic grading and has not been published elsewhere.

Stuttgart, 27. September 2008 Iñigo Ucero Aristu