CHALMERS
UNIVERSITY OF TECHNOLOGY

Detecting Network Degradation Using Machine Learning

Predicting abnormal network behavior with anomaly detection

Master's thesis in Computer Science – Algorithms, Languages and Logic

ADRIAN GASHI ROJAS
NICLAS OGERYD NORDHOLM

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2017

Master's thesis 2017:06

Detecting Network Degradation Using Machine Learning

Predicting abnormal network behavior with anomaly detection

Master's thesis in Computer Science – Algorithms, Languages and Logic

ADRIAN GASHI ROJAS
NICLAS OGERYD NORDHOLM

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden 2017

Detecting Network Degradation Using
Machine Learning
Predicting abnormal network behavior with anomaly detection
ADRIAN GASHI ROJAS
NICLAS OGERYD NORDHOLM

Supervisor: Harald Lüning, Ericsson AB
Alexander Schliep, CSE
Examiner: Richard Johansson, CSE

Master's Thesis 2017:10
Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2017

Detecting Network Degradation Using
Machine Learning
Predicting abnormal network behavior with anomaly detection
ADRIAN GASHI ROJAS
NICLAS OGERYD NORDHOLM
Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

Due to the prohibitive cost of downtime in large complex systems, it is important to reduce or entirely eliminate any downtime that might happen as a result of degradation in system quality.
This thesis paper presents a newly developed model for network congestion detection in large scale networks, using Gaussian Mixture Model and Isolation Forest algorithms to improve early detection of failure in the Ericsson EBM data. The results from the evaluation suggests that the developed method is more robust and precise in finding anomalies, compared to the current method used by Ericsson.

Keywords: machine learning, anomaly detection, packet loss, telecommunication, prediction

Acknowledgements

We would like to thank Ericsson for giving us the opportunity to work in such an interesting field, as well as our supervisor at Ericsson Harald Lüning.
We would also like to give a special thanks to our supervisor Alexander Schliep for coaching us through the whole thesis project, and providing us with expert knowledge in the subject.
We give our thanks to our examiner Richard Johansson.
Additionally, we would like to thank Håkan Tranberg at Ericsson for providing expert knowledge in the Ericsson EPC system.
Finally we would like to thank everyone else at Ericsson that has done their best to help us during the thesis.

Dedicated to my dear friend Niclas Ogeryd, who sadly passed away during the thesis work. May you rest in peace.

ADRIAN GASHI ROJAS, Gothenburg, Sweden 2017
NICLAS OGERYD NORDHOLM, Gothenburg, Sweden 2017

Contents

List of Figures

List of Tables

1

Introduction

The following sections provide background information about the purpose of the thesis, the problem description and presents the goals and challenges of this thesis.

1.1 Background

Due to the high cost that is incurred from downtime in large complex IT systems, one can easily see why predicting failures in real-time before they occur and applying preventive methods would be of interest [2]. In the Ericsson telecommunications system's core there are multiple network nodes involved that together are capable of handling millions of mobile broadband subscribers simultaneously. These nodes are responsible for functionality such as mobility, payload, charging, subscription handling and end user security.

The demands for availability on these nodes are extremely high. Nevertheless (partial) system restarts do occur for reasons spanning from anything from a process that crashes and restarts to an entire node restarting, causing a network outage. Since network outages can affect millions of mobile users, this is highly undesirable.

A node provides several real-time streams that are constantly supplying information about its status, e.g. for network monitoring and troubleshooting purposes. This data constitutes a huge amount of information including system logs, alarms, process monitoring, and Event Based Monitoring (EBM) data.

Since the system nodes are built for extremely high availability, there is a low probability of severe crashes occurring. What is more likely to happen is that the network quality decreases over time, due to network congestion or a functional problem with one or more nodes in the system, causing a performance degradation.

Although the system has many preventing measures and alarms that can facilitate a fast recovery if a problem would occur, the end users would still experience a performance degradation until the problem is solved. Another big problem for Ericsson is also that the available alarms in their system aren't that smart, causing a lot of false alarms and alarms that are hard to debug.

In order to improve the availability of the system, it is proposed that by using machine learning one could possibly be able to predict an upcoming network congestion before a performance degradation occurs, and also facilitate the debugging process.

When the latency of a signal is too high, a timeout occurs. When a timeout occurs,

the signal either does a retry or gets dropped. In the presence of traffic congestion, the latency of many signals would therefore increase, and some would reach timeout or get dropped. By analyzing the log data, this pattern of increasing latency and timeouts could be detected before the congestion becomes too severe, causing a network outage.

Since the amount of data is incredibly large, it is nearly impossible to detect any patterns manually. If those patterns could be detected in real-time by a machine, preventive measures could be applied to minimize the damage to the system.

1.2 Project Goals

This thesis was divided into three goals, visualized in Figure 1.1. The main goal was to develop and evaluate a method for online detection of increasing packet loss in a multi-node telecommunication system. The method had to be able to detect an increasing trend in a certain amount of time before a severe problem occurred, in order for preventive measures to be applied in time.

The second goal was for the prediction to also include a classification of the failure, to facilitate debugging. The classifications would indicate the nature of the congestion, e.g if a Denial Of Service attack was taking place, or if there's a failure in one of the system nodes that causes a bottleneck for the system.

Lastly, the third goal was for the prediction method to also perform a root cause analysis, which would identify the source of the problem, in order to further facilitate debugging.

The goal of using failure prediction was to be able to apply preventive measures and minimize damage on the system. However, this project was focused on performing failure predictions, not on applying countermeasures. Furthermore, although many different logs existed, the focus was on using the available EBM data from the MME node, which will be discussed in Section 3.1.2.

Due to complications throughout the project, only the first goal was implemented. This will be further discussed in this thesis. The second and third goal are proposed as work to be done. This is discussed in Section 5.4.

1.3 Related Work

Online failure prediction is a well-studied topic, and there exist several methods that suits different types of systems [3]. Recent work by Felix Salfner et al. on online failure prediction in telecommunication system explored the possibility of predicting failures in real-time by using semi-Markov chains combined with clustering [1]. Their work was to compare their own failure prediction method, Similar Events Prediction (SEP), with two other well-known prediction methods. Their results showed a promising improvement of the performance of the predictions, compared to other popular prediction methods, with a precision of 80%. Their model did however rely

on a full understanding of all error patterns that the model should be able to predict. If a new unknown form of error would occur, their model would not be able to predict it.

Recent work by Balaji et al. studied prediction of network congestion in wireless networks [4]. They analyzed TCP Ack flags and used a Multi Step Clustering method to identify abnormal levels of packet loss in their wireless network. The results look promising and show a good efficiency in identifying packet loss. However, the method is limited to analyzing TCP and UDP packets only.

1.4 Report Disposition

In Section 2, a thorough explanation of the basic theory needed to understand the foundation of the project is provided. In Section 3, the methods used in the development of the model are explained. In Section 4, the evaluation of the model, as well as the results are provided. Lastly, Section 5 discusses the results.

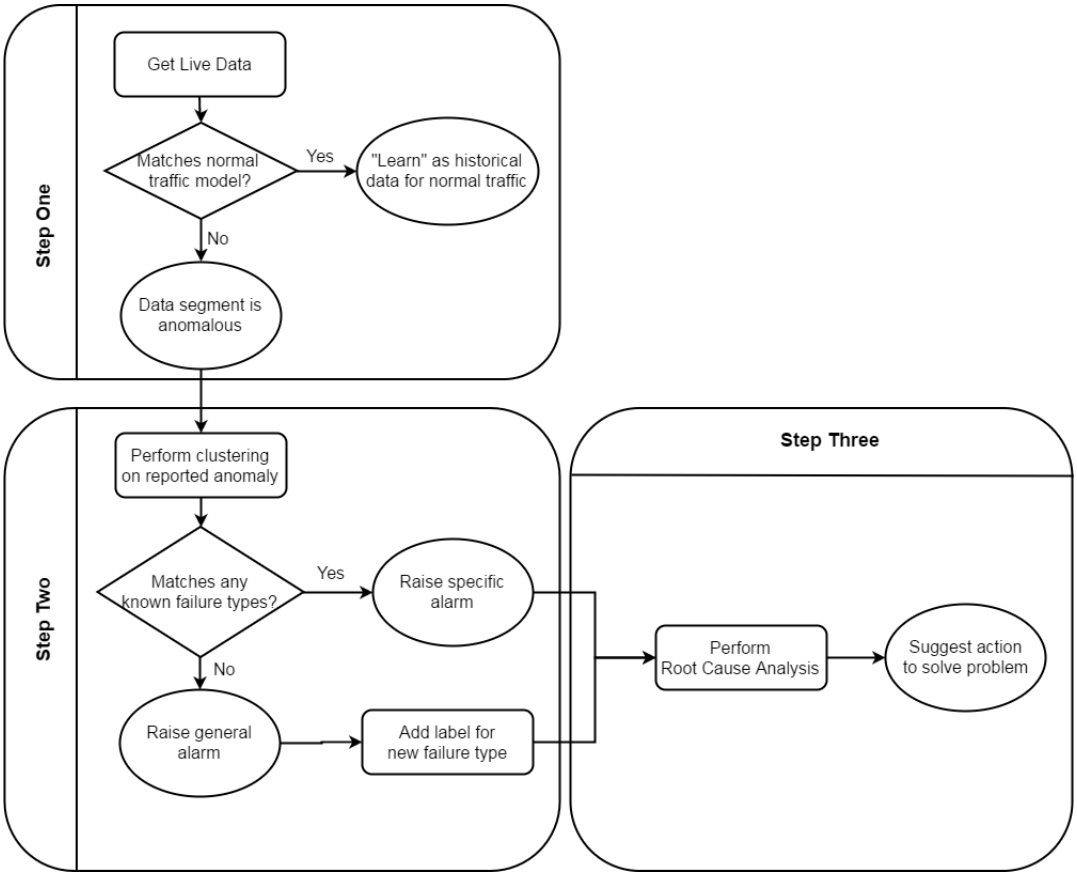

Figure 1.1: Visualization of the work flow and goals of this thesis

2

Theoretical Background

In this following section, detailed information needed to understand the rest of this thesis is presented. Firstly, basic information about machine learning is presented. Secondly, information about the algorithms used in this thesis is presented.

2.1 Machine Learning

Machine learning is a broad term that can be defined as the field of study where computers have the ability to learn without being explicitly programmed [5]. There exists many different machine learning algorithms, which can be divided into two major categories; unsupervised- and supervised learning, which are explained in the following sections [6]. It is well known that machine learning comes from, and has its roots, in statistical modeling.

Leo Breiman suggests that the data models used in statistical analysis are becoming more cumbersome due to increased complexity of the data used [7]. Instead, he proposes that a better solution would be to try more algorithmic approaches, where the model is a black box, created by finding some function $y = f(x)$, with y being the response to the input x. A basic overview what these two approaches looks like can be seen in Figure 2.1.

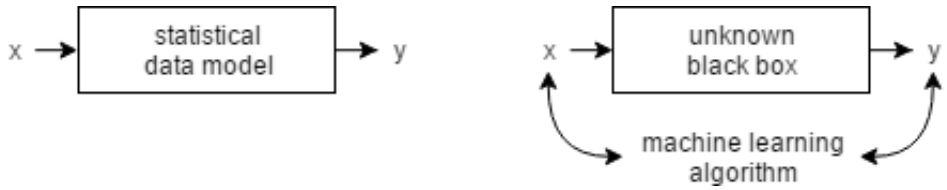

Figure 2.1: Simplified view of Broman's comparison between Data Modeling and Algorithmic Modeling

2.1.1 Supervised Learning

In supervised learning, the goal of the algorithm is to take a set of input vectors,$\mathbf{X}$, and their corresponding *labels*, $\mathbf{Y}$, in order learn the mapping between these values [8]. By doing so, the algorithm should be able to infer the labels for previously unseen values of $\mathbf{X}$. These models are trained by being fed the aforementioned list of feature vectors $X_i = \{x_{i1}, ..., x_{in}\}$ coupled with their corresponding output labels

$\mathbf{Y}_i$. The result of this is a decision function that should be able to label new values of X. Bishop [9] states that there's two different approaches, when dealing with discrete variables (*classification*) or continuous variables (*regression*):

- Classification:
  The objective of classification is to determine categories for data, based on its features. As the name implies, the decision function is split into 'classes', each of these being described by a combination of the before mentioned features. Examples of this could be medical research where the health status of individuals are classified by their test results or deciding the sub-species of a plant based on its phenotype.

- Regression:
  The field of regression is based on its namesake in statistical analysis. In machine learning the result of the regression analysis is used as a basis for the decision function. This function is then used to predict a real value from new input.

2.1.2 Unsupervised Learning

Unsupervised learning is used to identify a function that describes a hidden structure of some unlabeled data.

An unsupervised algorithm is given a training set which only contains a set of input vectors without any corresponding output values. Since it can't map the input values to the output values, its goal is instead to map the structure of the input data. This can then be used to find groups of similar data, called **clustering**, or to determine the distribution of data, known as **density estimation**.

2.2 Anomaly Detection

Anomaly detection is the method of identifying observations in a data set which do not conform to an expected pattern. This a common approach in many Network Intrusion Detection systems, because it doesn't rely on knowledge of all possible anomalous patterns and is therefore very robust in finding unknown types of anomalies [10]. Most methods for anomaly detection are based on unsupervised algorithms, mainly clustering. By clustering normal data points together, the algorithm can classify new data points as anomalous if they lie to far away from the normal cluster. This approach will be the main focus of this thesis.

2.3 Decision Tree Learning

A popular method for machine learning is decision trees, a predictive model that uses tree-like graphs, similar to a flowchart. Each internal node in these trees represents a test, with each child branching out from this internal node being one of the possible outcomes. The terminal nodes, or leaves, of the tree represents a label or

classification, that is the decision that is made from traversing the tree (hence the name decision trees. This type of model works almost "off-the shelf", since they're invariant when applying monotone transformations to the predictor data [11]. Another great benefit is the internal feature selection, where outliers are partitioned into small subspaces.

The decision tree works by performing a *recursive partitioning* in each step, where each predictor variable is compared and the 'best' choice is greedily selected. This selection is done based on purity of the partitions, that is how well each class is divided at each split, calculated through

$$purity(\Omega, \mathbb{C}) = \frac{1}{N} \sum_{k} \underset{j}{max} \, |\omega_k \cap c_j| \tag{2.1}$$

where **N** is the sample count, $\Omega = \omega_1, \omega_2, ..., \omega_k$ is the possible clusters, which in the case of decision trees takes the form of two partitions, and $\mathbb{C} = c_1, c_2, ..., c_j$ is the possible decision classes that the sample data belongs to [12]. Basically, the purity function gives a measure of how well each class is separated into its respective cluster.

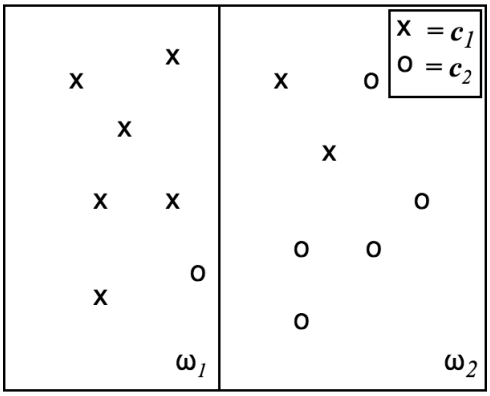

Figure 2.2: Example of partitioning used in purity calculation

Using the clustering of Figure 2.2 as an example, the purity would be

$$purity(\{\omega_1, \omega_2\}, \{c_1, c_1\}) = \frac{|\omega_1 \cap c_1| + |\omega_2 \cap c_2|}{N} = \frac{6+5}{14} \approx 0.79 \tag{2.2}$$

From this example one can see that when $k = N$, i.e. when the number of clusters and samples is equal, one would get a perfect purity score of 1 at the cost of a high generalization error.

This greedy approach, where the cluster partitioning performed in order to achieve the highest purity, ends up being a common problem in decision trees. Since the recursion only stops once the purity increase is small to none, the trees are prone to overfit on noise in the sample data as you get further from the root [13].

There's two major methods to deal with over-fitting in decision trees, namely *early stopping* and *post-pruning* [14]. The former works by using a statistical test to stop earlier and creates an external leaf at a point before the process would usually

end. In post-pruning the whole tree is created normally and the resulting tree is then trimmed down by removing sub-trees that doesn't have a big effect on the classification outcome.

2.3.1 Random Forest

One way to mitigate the over-fitting of decision trees is to use random forests where the prediction is made based on an ensemble of fully grown decision trees [15]. The decision is then made through majority vote, in the case of classification, or average value, in the case of regression.

There is a key difference however, instead of using the optimal split each of the trees makes the selection within a random sample of the predictors [16]. This means that the kth tree $h_k(\mathbf{x}, \Theta_k)$ is grown from a random subset Θ_k of the input variables, which, given the input $\mathbf{x}$, results in a forest of random tree classifiers

$$randomForest(\mathbf{x}) = \{h_1(\mathbf{x}, \Theta_1), \cdots, h_k(\mathbf{x}, \Theta_k)\} \tag{2.3}$$

The resulting value from this forest is the majority voted/average among the trees, as mentioned before. Due to the Law of Large Numbers this value is guaranteed to converge, meaning that the problem with over-fitting in regular decision trees is avoided [16].

2.3.2 Isolation Forest

Isolation Forest (iForest) is an unsupervised algorithm for outlier detection that uses random forests [17]. The approach of most existing models for anomaly detection is to learn a profile for normal data sets, and then comparing new data samples to the normal data in order to identify anomalous data points.

The method itself has a similar approach to random forests. An ensemble of fully grown decision trees is built through random sampling of the input predictors. Given that an outlier, by definition, is more likely to exist further away from other observations, one can use this distance to determine how 'abnormal' an observation is [18]. An example of this can be seen in Figure 2.3 where it's the anomaly is easily partitioned, resulting in a shorter path length in the tree.

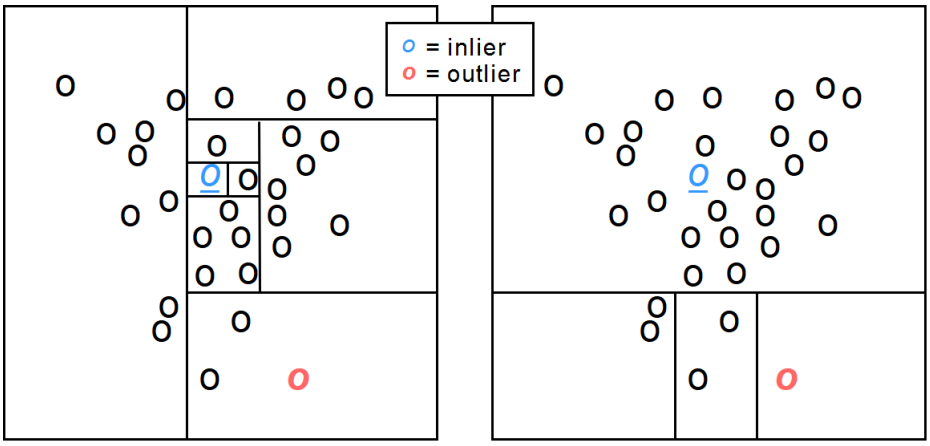

Figure 2.3: Example of partitioning in an isolation forest

Given that a tree in the iForest is a *proper binary tree*, it has the same properties as a Binary Search Tree (BTS) [17] . This means that the average path length of a decision tree is equal to an unsuccessful search in the BTS, which gives you

$$c(n) = 2ln(n-1) + \gamma - 2\frac{n-1}{n}, \tag{2.4}$$

where n is the size of the input data set and γ is Euler's constant. This average length, $c(n)$, is then used in order to get a normalized path length used in the calculation of the anomaly score

$$s(x,n) = 2^{-\frac{\hat{h}(x)}{c(n)}} \tag{2.5}$$

where x is the current sample in a data set of n samples and $\hat{h}(x)$ is the number of partitions needed in order to isolate x. Based on Equation 2.5, it can be seen that for low values of $\hat{\mathbf{h}}(x)$, the anomaly score trends towards 1. This can be interpreted as x having a high likelihood of being an anomaly, since, on average it's easily partitioned in the iForest.

$$s = \begin{cases} 0, & \hat{h}(x) \to (n-1) \\ 1, & \hat{h}(x) \to 0 \end{cases} \tag{2.6}$$

Since a recursive partitioning can be represented by a tree structure, the number of splits that are required to isolate an observation is equivalent to the path length from the root node to the final node. The path length from all random trees can then be used as a measure of abnormality. Anomalous data points are highly likely to produce noticeably shorter paths in a random partitioning. [19].

One of the advantages of the iForest model is that it's well suited for large and high-dimensional data, due to the lack of distance and/or density calculations that many other methods use, which are heavy in computational cost.

2.4 Gaussian Mixture Model

A Gaussian Mixture Model (GMM) is a probabilistic model that is used to identify a number of Gaussian distributions, or components, with unknown parameters in a given data set. The model uses a generalized k-means clustering method to identify the covariance structure of the data as well as the means of the latent Gaussian components. The algorithm definition can be shown by:

$$p(x) = \sum_{i=1}^{K} \Phi_i N(x|\mu_i, \sigma_i) \tag{2.7}$$

$$N(x|\mu_i, \sigma_i) = \frac{1}{\sigma_i\sqrt{2\pi}} exp\left(-\frac{(x-\mu_i)^2}{2\sigma_i^2}\right) \tag{2.8}$$

$$\sum_{i=1}^{K} \phi_i = 1 \tag{2.9}$$

For a Gaussian mixture model with K components, the kth component has a weight of ϕ_k, mean of μ_k and a variance of σ_k.

The Scikit-learn library contains different implementations of GMM, that each correspond to a different estimation strategy. Two of these implementations are Gaussian Mixture and Variational Bayesian Gaussian Mixture.

2.4.1 Gaussian Mixture

The Gaussian Mixture implementation uses the estimation algorithm **expectation-maximization (EM)**. EM is a well-founded statistical algorithm that is used to compute the probability that a given point belongs to each component of the model. The algorithm iterates over the parameters in order to maximize the likelihood of the data belonging to a component. As the name implies, the EM algorithm cycles between the two steps Expectation & Maximization until it converges to a local optimum.

In each iteration, the EM algorithm starts by calculating the expected lower bound shown in Equation 2.10. This is done by using a set of observations, X, the latent variables, Z, and the current parameter expectations, Θ^t

$$Q\left(\Theta|\Theta^t\right) = E_{p(Z|x,\Theta^t)}\left[log\, P(X, Z|\Theta)\right] \tag{2.10}$$

This is then used as a basis for the maximization part, where the parameter estimation for the next time step ,Θ^{t+1}, is calculated by maximizing the new parameters Θ^{t+1} as shown in Equation 2.11. As mentioned before, this process is repeated until it converges, which is guaranteed to reach a local maximum. [20][21]

$$\Theta^{t+1} = arg\, max_{\Theta}\, Q(\Theta|\Theta^t) \tag{2.11}$$

One advantage of using the Expectation-Maximization method to estimate a Gaussian Mixture is that it's one of the fastest algorithms for learning mixture models. However, if there are insufficiently many points in a mixture, it has difficulties in estimating the covariance matrices. This causes the algorithm to diverge and find solutions with infinite likelihood.

2.4.2 Variational Bayesian Gaussian Mixture

The Bayesian Gaussian Mixture implementation uses the estimation algorithm **Variational Inference (VI)**, which is an extension of the expectation-maximization algorithm [22]. VI maximizes a lower bound on model evidence instead of data likelihood as EM does.

The process is the same as for EM, where it iterates through the probability for each point belonging to a mixture component. However, it also adds regularization by integrating information from previous distributions. This avoids the problem that is often found in EM, where it might diverge and find infinite likelihood. It does however also introduce some subtle biases to the model.

2.5 Online Failure Prediction

The goal of online failure prediction is to, in real-time, forecast if a failure will occur in the system in the nearest future. Compared to reliability prediction in software, that is made offline, the online predictions depends on the current state of the system and handles shorter time intervals. The concept is visualized in Figure 2.4. A prediction made at time t should predict if at time $t + \Delta t_1$ a failure is going to occur or not. Δt_1, which is called *lead time*, has a lower-bound Δt_w which is called *warning time*. The warning time refers to the time it takes for the system to restart or perform preventive measures if a failure is predicted. Therefore, for the prediction to be of any use, the lead time has to be at least as long as the warning time. But if Δt_1 is too large, the predictions will loose accuracy. The period of the prediction, Δt_p, refers to the time for which a prediction holds. A large Δt_p increases the chance that a failure occurs within the period. However, if Δt_p is too large, the prediction loses its usefulness.

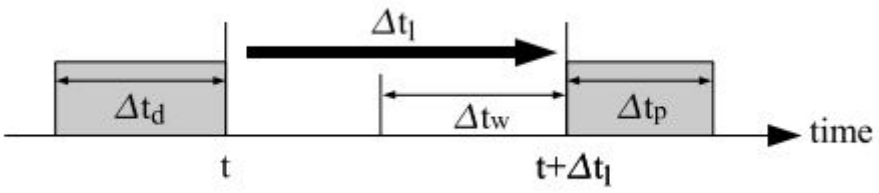

Figure 2.4: Definition of online failure prediction [1]

3

Implementation

The following sections present the different tools that were used, background information about the Ericsson telecommunication system and how the developed model was implemented.

3.1 System background

The Ericsson telecommunication system has a highly complex architecture with numerous nodes collaborating in order to provide their service to millions of mobile users. The system consists of many areas that provide different services, but the area in focus for this thesis is the Evolved Packet Core.

3.1.1 Evolved Packet Core

Evolved Packet Core (EPC) is the core network of the Long Term Evolution (LTE) system, and is the latest evolution in the 3GPP core network architecture . EPC is capable of providing millions of users with mobile packet data services. The network consists of a number of components, or nodes, which handles different tasks in the system. An overview of the architecture of EPC, with all of its major nodes, is shown in Fig 3.1. The main nodes of the system are the Serving Gateway (SGW), the PDN Gateway (PGW) and the Mobility Management Entity (MME).

The SGW node handles all the incoming IP packets, and serves as a local mobility anchor for when the users move between different mobile zones. It also performs some administrative functions, such as collecting information for charging (e.g the volume of data that a user sends or receives).

The PGW node is responsible for IP address allocation, as well as Quality of Service enforcement for guaranteed bit rate. It is also responsible for the filtering of downlink user IP packets.

The MME is in charge of processing the signals between the users and the network and is the node of interest in this thesis. The node handles the mobility of all the users, and routes the traffic to the right destination in the network. The MME consist of multiple identical physical cards that cooperate in handling the incoming traffic. There is a balancing algorithm in place to control that all the cards handle

the same amount of traffic, to make sure that all cards are used efficiently. The node is highly scalable and can handle up to 36 million users.

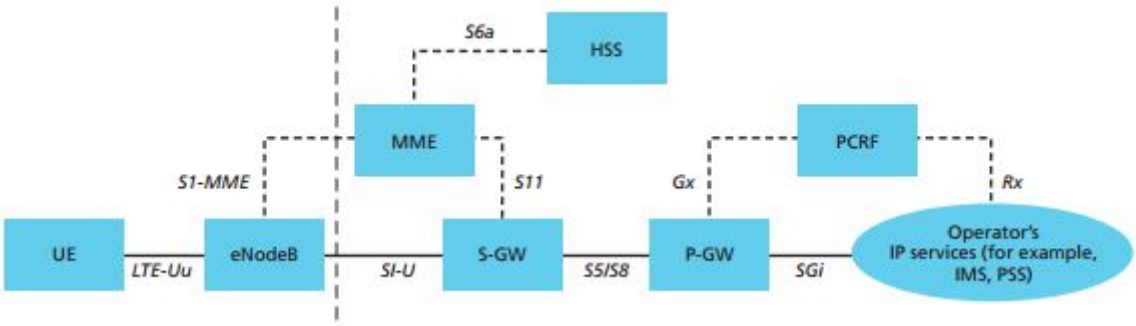

Figure 3.1: Overview of the EPC network architecture

3.1.2 Event Based Monitoring

EBM is a flexible performance monitoring solution for real-time event provisioning, which logs successful and unsuccessful events for completed mobility and session management procedures. The format itself is proprietary to Ericsson. However, it maps 1:1 to the GTPv2 (GPRS Tunneling Protocol v2) standard [23]. This is specified by the 3rd Generation Partnership Project (3GPP), which is an international partnership that unites several telecom development organizations [24].

A MME node generates an EBM event for every incoming signal that it processes. The EBM data stream is stored in a binary format in a configurable time interval. Each event consists of approximately 70 features of detailed information about the nature of the event between an User Equipment (UE) and the MME node, e.g when a session is created, deleted or if data packets are sent. The events contain information such as event type, time stamps, round trip time of the message, cause codes that describe what caused the event to happen, the ID of the user and more. For this thesis the duration parameter, which represents the round trip time for an event, is the parameter that is mainly used. An example of how the EBM data looks like is shown in Figure 3.2

RAT	Event Type	Min [ms]	Max [ms]	Average [ms]
GSM	ATTACH	0	54729	3237
GSM	ISRAU	0	32410	2777
WCDMA	SERVICE_REQUEST(PAGING)	4	16903	2676
GSM	RAU	0	65535	1924
WCDMA	ATTACH	0	65535	1773
WCDMA	ISRAU	0	65535	992
WCDMA	ACTIVATE	0	29033	918
GSM	DETACH	0	30018	582
WCDMA	RAU	0	30019	393
WCDMA	DEACTIVATE	0	40027	309
GSM	DEACTIVATE	0	20052	251
WCDMA	SERVICE_REQUEST(NO PAGING)	0	14784	245
GSM	ACTIVATE	0	29045	109
WCDMA	DETACH	0	9016	30

Figure 3.2: Example of the EBM duration data when analyzed by EBM Log Tool 3.3.

By analyzing the EBM data streams, one could find patterns in the events that indicate that the performance of the system might be deteriorating. If for example

the duration parameter of an event suddenly spiked above normal levels, this might indicate that a connection between nodes is congested or failing. By using the cause codes that are included in the EBM data, one could do further examination in order to find the source of the problem.

3.2 Data Preprocessing

An important part of machine learning is the process of cleaning the data and parsing it to a format that the machine learning model can use. Since the EBM data is stored in a binary format, it had to be parsed to a readable format before being passed to the model. A proprietary Perl script was used to achieve this, which created a text file with parsed data in CSV-format. Since the data consists of many features that are not necessary for this work, the appropriate features had to be extracted before passing the data to the model.

The loading of the data was done by using the Pandas library, which is a high performance python library for data analysis [25]. In order to speed up the loading process, the duration parameters could be explicitly specified to be extracted, thus ignoring all other parameters.

Ericsson has a large database of stored EBM logs that was used to study the structure of the data. There was however no data from a scenario where packet loss increased over time available. A simple data simulator was therefore created in order create a data set that could be used for evaluation. The simulator takes a given mean, variance and probability of packet loss and draws random data points with a normal distribution that represents the latency of events from a simulated traffic scenario. The data that was created for evaluation is further described in Section 4.2.

3.3 Tools and Frameworks

A proprietary tool, called EBMLogTool, was used to get statistical summary of all the different events in the EBM data. It could provide a detailed description of the average behaviour of each event type. This made it easier to analyze the different parameters, which would be used as features in the model. The average values observed in the summary were also used to get a baseline value that would later be used in the simulated data, described in Section 4.2.

For the development of the model, the platform **Anaconda** was used, which is a high performance distribution of data science packages for both Python and R. The platform provides access to hundreds of packages, but for this thesis the Python library **scikit-learn** was mainly used. The scikit-learn library is a simple and efficient open source library that contains a wide range of powerful machine learning algorithms [26]. Its simple yet powerful API makes it a great choice for this kind of machine learning problem.

3.4 Model development

After analyzing the EBM latency data, one could see that the behavior of the data confined to a finite number of Gaussian components. Furthermore, one could see that the latency data had a high variance, and a small appearance of extreme values. Using only the raw latency data as input to the Isolation Forest was concluded to not be feasible, since the high spread of the data made it difficult for the Isolation Forest to properly identify true anomalies. This information lead to the decision of using GMM in order to identify the Gaussian components and transform the raw latency data into a fixed number of classes, making it easier for the Isolation Forest algorithm to cluster the data and find anomalies.

Development of the anomaly detection model was separated into two mayor modules, one for each algorithm that was used, Isolation Forest (iForest) and Gaussian Mixture Model (GMM), described in Section 2.3.2 and 2.4 respectively. The model went through two phases during development, training phase and testing phase. An overview of the model is shown in Fig 3.3

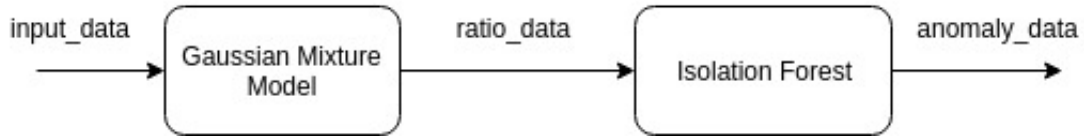

Figure 3.3: Overview of the anomaly detection model

3.4.1 Training Phase

In the training phase, GMM was fed a two-dimensional array of training data, consisting of event latencies where each row has one column with an integer value that represented the latency sample of one EBM event. The GMM then identified the parameters of the Gaussian components existing in the training data. The same training data was then fed to the GMM again for prediction, which returned a one-dimensional array of classifications for each sample, as shown in Equation 3.1.

$$Classification = \begin{cases} 0, & First\ Gaussian\ Component \\ 1, & Second\ Gaussian\ Component \\ 2, & Third\ Gaussian\ Component \end{cases} \tag{3.1}$$

Using the list of classifications returned from the GMM, ratios of how the data laid between the components were then extracted by using a rolling window method on the classification data. This method is used to smoothen out irregularities in the data and create a new data set of ratio data based on the classification data.

The rolling window method works as the name suggests. It takes a subset of x samples from the data and performs an operation on that subset, which in this case was the operation in which the GMM for each sample predicts which Gaussian distribution the sample belongs to. It then rolls the window with a step k, thus removing last k samples and adding k new samples to the window. This process is

iterated throughout the whole data set, and produces a new data set of smoothed data. This is visualized in Figure 3.4

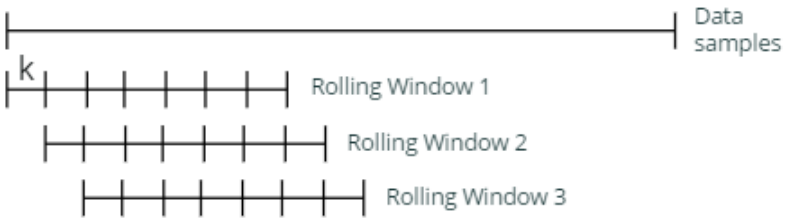

Figure 3.4: Illustration of the rolling window method

For each window, the data subset was fed to the GMM, which returned a prediction of which distribution each sample belongs to. The ratio between the components in which the data lays could then be calculated. An example of how the ratio data looks after the rolling window method is applied is shown in Table 3.1. Last step of the training phase was to feed the resulting ratio data from the GMM as training data to iForest, which does a clustering of the data in order to learn the parameters of the cluster of the normal data.

3.4.2 Testing Phase

After the training phase was completed, the testing phase began. Firstly, the test data set was fed to the trained GMM using the same rolling window method as described in the training phase, and new ratio data was extracted for the test data. The ratio data could then be fed to the iForest module for prediction, which returned a list of anomaly predictions for each sample, as Equation 3.2 shows. The resulting list of anomalies could then be used to evaluate the method. The method of the evaluation is further described in the next chapter.

$$Anomaly\ classification = \begin{cases} 0, & Normal\ data \\ 1, & Anomaly \end{cases} \tag{3.2}$$

Table 3.1: Example of ratio data from the GMM

sample	Component 1	Component 2	Component 3
1	0.95	0.04	0.01
2	0.92	0.05	0.03
3	0.93	0.06	0.01
4	0.94	0.04	0.02
5	0.91	0.07	0.02

4

Evaluation

The developed model was evaluated in two steps. Firstly, the precision of the classifications from the GMM module was evaluated. Secondly, the precision of the iForest module was evaluated by comparing it to the currently used moving average method applied by Ericsson.

In the following sections the experimental setup used for the evaluations is presented. The evaluation metrics and the result of the evaluations are also presented.

4.1 Evaluation metrics

Precision, recall and F-measure are evaluation metrics that are commonly used to measure the quality of classifications [27].

Precision is the ratio between the number of correctly identified anomalies and total number of predicted anomalies

$$precision = \frac{true\ positives}{true\ positives + false\ positives} \tag{4.1}$$

Recall is the ratio between the number of correctly identified anomalies and the total number of true anomalies

$$recall = \frac{true\ positives}{true\ positives + false\ negatives} \tag{4.2}$$

F-measure is the harmonic mean of precision and recall, and is approximately the average of the two

$$recall = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{4.3}$$

4.2 Experimental setup

In order to evaluate the developed model, data was generated to simulate a scenario where an increase of packet loss could be detected by the anomaly detection model.

The evaluation of the developed model was then compared to the current naive method of moving averages that is used by Ericsson.

With the help of one of Ericsson's experts and some manual observations of EBM data from a MME-node with normal traffic, a test case scenario could be constructed using the following assumptions:

- The latency of the EBM events conform to a normal distribution.
- When a packet is lost, a configured timeout value is added to the latency, representing a retry in sending.
- A packet performs a maximum of 2 retries before giving up.
- Normal traffic scenarios have a packet loss of approximately 5%.
- The system is considered to be in failure state when the packet loss is above 8%.

With the help of these assumptions data could be created to evaluate the model. Firstly, training data was created by generating 100 000 normally distributed random samples, with each sample representing the latency of an event. The training data was then injected with a constant 5% packet loss, which is considered a normal and accepted level of packet loss. An occurrence of a lost packet was simulated by adding the timeout value to a random generated sample, thus simulating a connection retry after a failed attempt. A second retry was also simulated on 1% of the data samples, by adding 2 timeout values to the generated sample, simulating two connection retries. The Gaussian distributions of the training data are shown in Figure 4.1.

Secondly, testing data was created which consisted of 20000 samples of normal data, which was similar to the training data, with a constant 5% packet loss. This was combined with 30000 samples of 'bad' data, which had an increasing packet loss over time. To simulate the increase of packet loss, the probability of a packet loss happening was increased every time a packet loss occurred. In order to make the data more realistic, some extreme outliers were also injected with a 1% probability. The test data is shown in Figure 4.2.

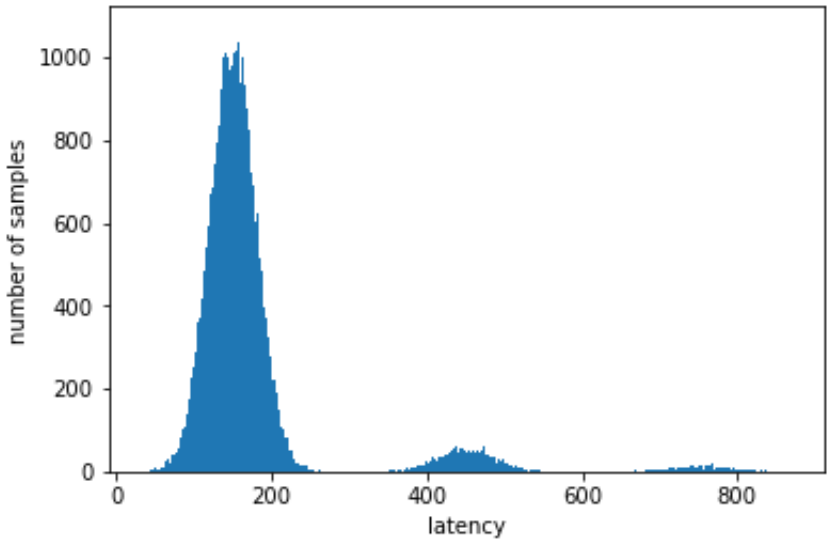

Figure 4.1: Histogram of simulated training data. The first Gaussian distribution represents normal data and the other two distributions represent packets that timed out at least once.

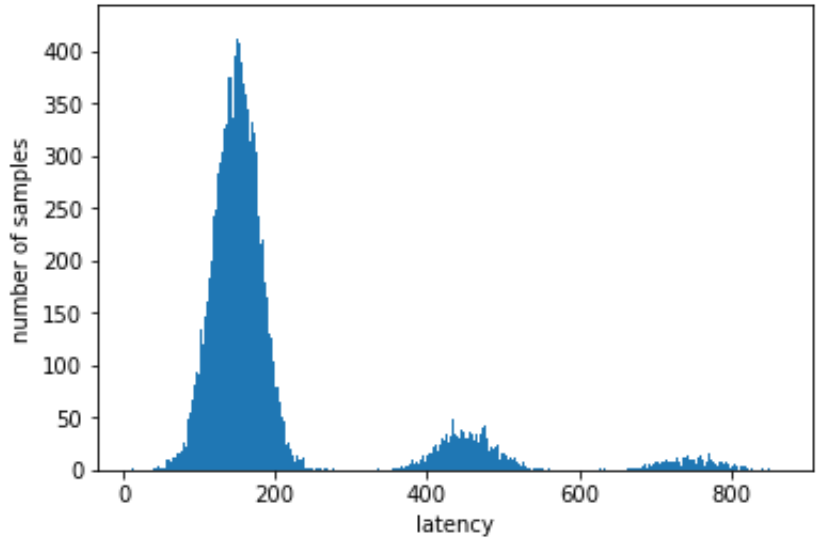

Figure 4.2: Histogram of simulated test data with increased packet loss

The generated data sets were then used to evaluate the model. The method of the evaluation is further explained in the following sections.

4.3 Gaussian Mixture Model Evaluation

The goal of the GMM module was to predict which Gaussian component each latency sample belonged to, so ratios between the found components could be calculated. In order to evaluate the accuracy of the GMM, the generated test data was labeled with either 0, 1 or 2, depending on if the sample was drawn from the normal data

distribution or one of the timeout distributions. The labeled data could then be compared to what labels the GMM predicted for the same data.

The result shows that the GMM does an excellent job in classifying the samples, with an accuracy of 100%. Figure 4.3 shows how the GMM has divided the data points into different classes with different colors. This great result does however depend on the value of the timeout, which in this case is quite distinct. This will be further discussed in Chapter 5

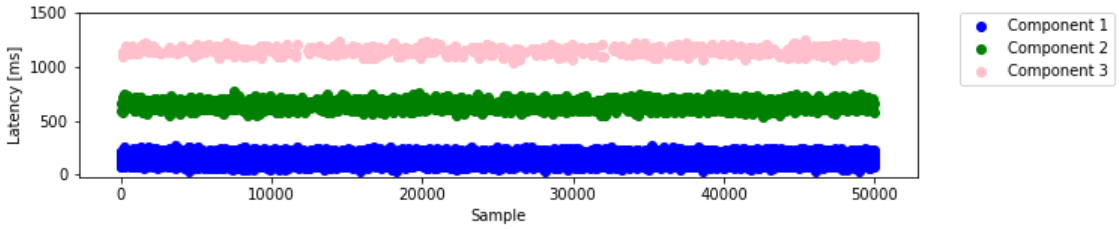

Figure 4.3: Result of GMM classification test

A big question when choosing this approach was if it would perform better then the moving average method in cases where there are extreme outliers present in the data, in regards to sensitivity. In order to evaluate the sensitivity of both approaches the generated training data was used. The data went through the rolling window method and ratio data and average data were extracted. The same process was done once again, but this time extreme outliers were injected into the training data first. Figure 4.4 shows the resulting data sets. The packet loss ratio on the y axis of the bottom plots represents the ratio of data samples that were not classified to be in the first Gaussian, meaning that these data samples were lost and did either one or two retries.

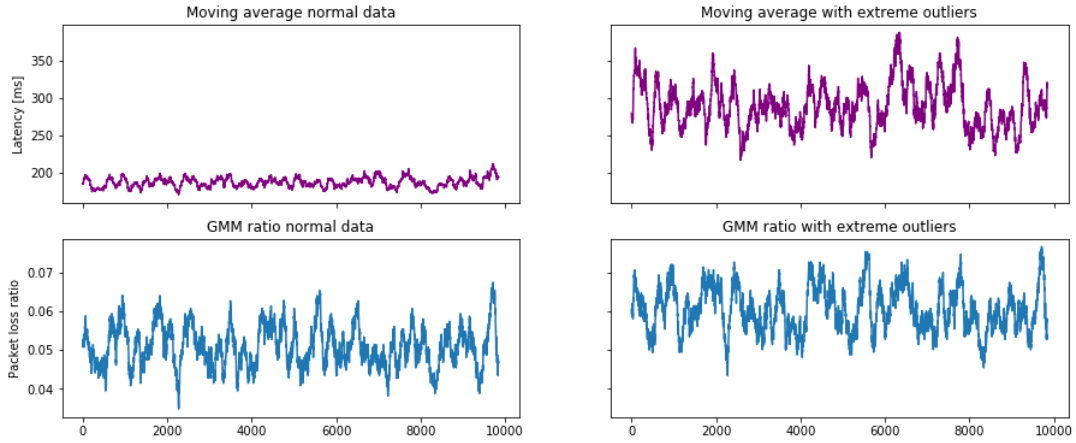

Figure 4.4: Result of sensitivity test

Firstly, the mean value was calculated for each data set. Secondly, the percentage increase of the mean value from the normal data set to the data set with extreme outliers was calculated. The resulting means and the increase are shown in Table 4.1

Table 4.1: Mean values of each data set, taken from the sensitivity test

	GMM ratio	Moving Average
Normal data	0.050815	186.818
Data with outliers	0.060459	288.932
Mean value increase	18.97%	54.66%

4.4 Model Evaluation

In order to evaluate the developed model properly, a comparison against the current method of finding anomalies had to be done. As mentioned before, the current method applied by Ericsson is a naive method where they monitor the average latency over time and manually set a threshold for which a latency above this threshold is considered an anomaly. So to implement this method in this case, the generated train and test data was put through a rolling window method where the average latency of each window was calculated, creating new data sets of average latency's. The data created from the training data was used to find a threshold that would be used to find anomalies in the test data.

One of the assumptions made in Section 4.2 was that the system is considered to be in a failure state when the packet loss is above 8%. This assumption was used to calculate the point in the data set where the packet loss crossed this limit and continued increasing over time. That limit was found approximately after sample 24000 in the test data. This limit was used to measure the precision and recall of the compared methods by assuming that all anomalies found after the limit were true positives, and anomalies found before the limit were false positives. The normal samples that occur before the limit are true negatives and the ones that occurs after the limit are false negatives.

The sensitivity of both models depend on the window size used in the rolling window method. Therefore, different window sizes were tested from the range 500-2000, and precision scores were calculated for each method with each tested window size.

Firstly, the model was trained using the generated training data from the experimental setup. The GMM learned the parameters of the normal data and ratio data could then be extracted from the module. The ratio data with window size 1000 is shown in Figure 4.5. The ratio data was then fed to the iForest for training.

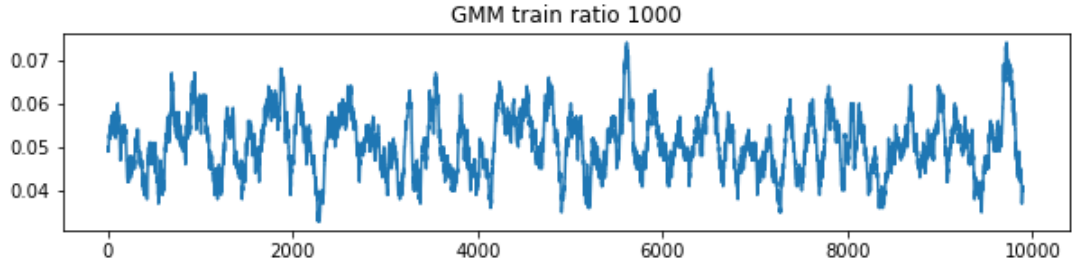

Figure 4.5: Ratio data extracted from the GMM training, with a window size of 1000 samples.

Secondly, the generated test data with increasing packet loss was fed to the model and the output returned was a list of anomaly predictions for all samples. The ratio data extracted from the GMM for the test data with window size 1000 is shown in Figure 4.6. The anomaly predictions made by iForest is shown in Figure 4.7

Each anomaly found in the prediction was then checked, to see if it occurred before or after the specified time limit of failure. A precision and recall score could then be calculated using Equation 4.1 and 4.2.

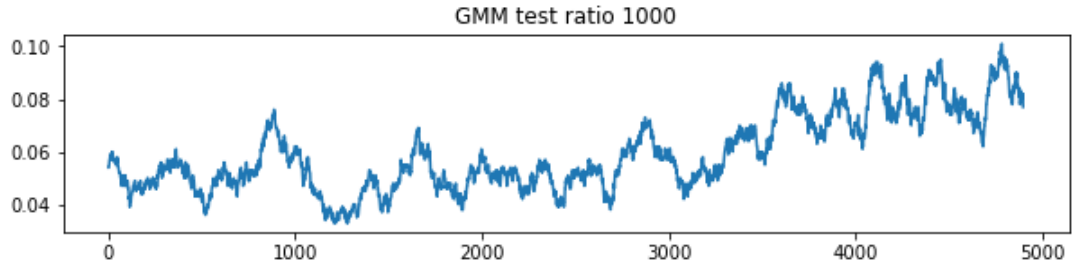

Figure 4.6: Ratio data extracted from the GMM evaluation test, with a window size of 1000 samples.

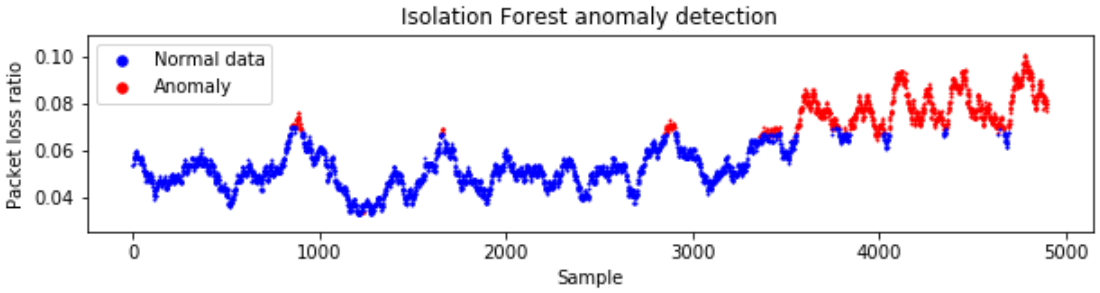

Figure 4.7: Anomaly prediction produced by Isolation Forest, using test ratio data

The same process was then performed with the moving average method. When applied on the training data, a new data set of averaged data was produced. The averaged data set from the test with window size 1000 is shown in Figure 4.8. A threshold was then manually specified by observing the training data.

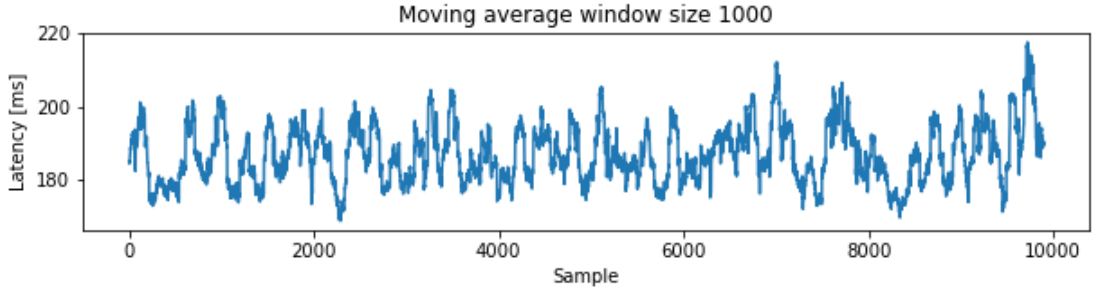

Figure 4.8: Data produced by the moving average method using training data, with a window size of 1000 samples.

The moving average method was then applied to the test data, shown in Figure 4.9 for window size 1000. Each sample that was above the specified threshold observed

from the training data was considered an anomaly. As was done with the iForest data, if an anomaly occurred after the specified time limit of failure it was considered a true positive, otherwise it was considered a false positive, vice versa with the normal samples being true negatives or false negatives. A precision and recall score could then be calculated and compared to the scores calculated from the developed model.

This process was done with both methods for each window size tested. The tested window sizes and the resulting scores for each method are presented in Table 4.2.

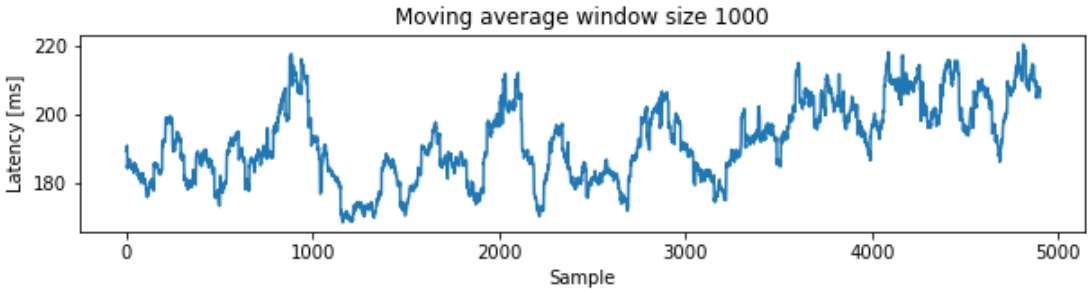

Figure 4.9: Data produced by the moving average method using test data, with a window size of 1000 samples.

Table 4.2: Precision and recall scores of Moving Average (MA) and Isolation Forest (iF), using different window sizes.

Window Size	MA Prec(%)	MA Recall (%)	iF Prec (%)	iF Recall (%)
500	80.8	9.1	94.4	40.2
1000	80.4	22.6	95.9	60.3
1500	80.3	44.5	87.8	73.6
2000	81.3	69.6	88.7	82.7

5

Discussion

The following sections discusses the results from the thesis, the method used and the ethical aspects of the thesis. Proposals for future work are also mentioned.

5.1 Result

The results from the evaluation test indicate that the proposed method of using Gaussian Mixture Models to classify data, and Isolation Forest to identify anomalies, is more robust and better suited for finding anomalies than the currently used method with moving averages. This is mainly due to the fact that the moving average method is very sensitive to extreme outliers, which occurs frequently in the EBM latency data. The proposed method is more robust because it normalizes the data, making it less sensitive to said extreme outliers, as could be seen in the evaluation Figure 4.4.

The evaluation tests performed in Section 4.3 were performed to evaluate if Gaussian Mixture Models could be used to classify the EBM latency data, and also if it would be less sensitive to extreme outliers than the moving average method. The result from the test showed that the introduction of extreme outliers to the data had lesser effect on the ratio data from the GMM compared to the data from the moving average method. This has to do with the fact that the GMM method has a fixed set of components, where the extreme outliers would end up in the same component class no matter how extreme they are. Compare this to the moving average, where a value that is several magnitudes larger than usual would severely pollute the average.

The GMM method only depends on the amount of outliers that are identified, not on the actual value of the data points. This is what makes it less sensitive to few extreme outliers. Whereas the moving average method is greatly affected by few extreme outliers, since a large value on one point can increase the average of a window a lot.

The result from the test of the accuracy of the GMM classification was perfect for the generated data set. When the Gaussian components in the data set are as well separated as they are in this data set, the GMM does a perfect job in classifying the data points to the correct component.

These results might however not be as great as conducted tests show, due to the fact that simulated data was used instead of real data. As mentioned before, there didn't exist any EBM data of known issues in regards to latency and network congestion.

Therefore, the simulated data was created with the help of Ericsson experts to model a somewhat real scenario. The complexity of the Ericsson network does however make the issue more complex than what has been modeled in the simulated data.

Furthermore, some of the assumptions made for the simulated data and for the method might not be entirely accurate. The data is assumed to be normally distributed, which might not be true in all cases. This assumption was based on an observation from one hour of real live data from a MME node. Furthermore, the Gaussian components are assumed to be well separated by a timeout value, which makes it fairly easy for the GMM to classify the data. The is quite simplified compared to real data where there might be many polluted data points in between the components. This would make it harder for the GMM to classify the data points correctly, compared to this case where it scored 100% in accuracy. However, since the assumptions are based on real observations and expert knowledge, it's considered to be a good starting point for further development of anomaly detection in large scale networks.

5.2 Method

An important part of the performed method was to use the rolling window method to smooth out the latency data. The result from this method is greatly affected by the window size used, since a larger window can smooth out the data too much, thus producing a non-sensitive new data set. This would result in anomalies possibly not being detected until the problem has become too large and it's too late for countermeasure to be applied.

On the other hand, a small window size can produce a data set that is highly sensitive to few extreme outliers. This would result in a huge amount of false alarms being produced by the model. Finding the perfect window size proved itself to be challenging, and more tests would have to be conducted to establish the correct answer to this problem.

As mentioned before, using GMM for classifying data might not be the most optimal approach for this task. The main problem with the approach is that the data is assumed to be normally distributed, which might not be the case for real data. In order to find the best approach, one would have to examine live data over an extended period of time to conclude that the behavior of the data is indeed normally distributed.

5.3 Ethical aspects

There are some ethical issues when working with this type of data. The EBM data contains detailed information about the signaling from users to the network, which means that the behavior and location of a user can easily be tracked through analysis of the EBM data. This was a major concern which resulted in no data being available for analyzing. In order to acquire data from live nodes, all the sensitive information would have to be censored before it could be analyzed by any machine learning

model. There does exist some processes for censoring the EBM data automatically, but it has to be manually supervised to make sure that no sensitive information is leaked. This is obviously very time consuming, due to the sheer amount of data that is produced, and causes a lot of trouble when performing this type of experiment.

5.4 Future Work

The work that has been done in this thesis has made ground for further work in the area of anomaly detection in telecommunication systems. What needs to be done in future work is to firstly make sure to have the appropriate data available. Working with real live data is essential for a proper evaluation of the method that has been used. It would also be necessary to collect data from known cases where network congestion has caused a performance degradation on the network. This data is needed for analysis, in order to learn the correct behavior of the system when network congestion occurs.

For future development, using more of the available information in the EBM data would be preferred. This thesis focused on its latency parameter due to lack of real data, which ended up stripping away all the complexity of the problem by only using one feature. Many other features in the data, such as cause code and event result, may contribute greatly to a better prediction by the model. Using these might also make way for the prediction of other types of problems in the network.

Another feature that needs more work is to make sure that the anomaly detection model can analyze the EBM data and detect anomalies in real-time. Some investigation will have to be made on how to stream the EBM data in real-time, since as for now the data is sent out in 15 minute intervals by default. The goal was to use this feature in the thesis, however due to the problems with acquiring the data this was not feasible.

The method of using GMM would also need to be further evaluated, before starting any extensive development using it. In this thesis, it was assumed that the latency data had a well separated timout intervals timeou, which created a perfect scenario for the GMM. Since this might not be the case when dealing real live data, another approach might be required. Especially if more features from the EBM data were to be used, in which case a more complex classification method would be needed.

Due to the EBM data being such a complex and feature-rich format, it's possible that a neural network would be helpful in order to find some hidden patterns. One potential area is root cause analysis, where the neural network could possibly identify the source such as a distant link failure or another network node going down.

6

Conclusion

When looking back at the goals that were set for this thesis, which are described in Section 1.2, one can see that there were some shortcomings in the method. The first goals, which was to develop a packet loss detection model, could be considered complete although not entirely optimal. A first prototype of a detection model was developed and evaluated, which could be viewed at as a success.

However, due to the delays caused by not having the proper data needed for the development, the second and third goal were never implemented. We are however confident that it would be possible to implement both, if more data and time were available.

Due to the lack of the aforementioned data, a lot of time had to be spent trying to build a simulation model that reflects reality. Obviously there are some major problems with this approach. The foremost being that, due to the sensitivity and size of the data, there's no EBM-records containing these errors happening in a system. Instead they are modelled based on descriptions that we acquired from an expert in the subject matter. This time spent on building a data model could have been spent on other more important things, such as development of the detection model.

Furthermore, since no proper data was available, trying to find the correct method for the job was a bit challenging. In order to progress in the project, many assumptions had to be made. This could have been done a lot differently if the proper data was available, and the result of the thesis could have been a lot different.

Most of the problems faced in this thesis are all based on the fact that real live data wasn't available. This is of course very unfortunate, but the best was done with the cards that were dealt, and the resulting work is considered a good foundation for future work in this area.

Bibliography

[1] F. Salfner, M. Schieschke, and M. Malek, "Predicting failures of computer systems: A case study for a telecommunication system," in *Proceedings 20th IEEE International Parallel & Distributed Processing Symposium*, pp. 8–pp, IEEE, 2006.

[2] M. Shatnawi and M. Hefeeda, "Real-time failure prediction in online services," in *2015 IEEE Conference on Computer Communications (INFOCOM)*, pp. 1391–1399, April 2015.

[3] F. Salfner, M. Lenk, and M. Malek, "A survey of online failure prediction methods," *ACM Computing Surveys (CSUR)*, vol. 42, no. 3, p. 10, 2010.

[4] V. Balaji and V. DURAISAMY, "Cluster based packet loss prediction using tcp ack packets in wireless network,"

[5] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM J. Res. Dev.*, vol. 3, pp. 210–229, July 1959.

[6] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 ed., 2003.

[7] L. Breiman, "Statistical modeling: The two cultures (with comments and a rejoinder by the author)," *Statist. Sci.*, vol. 16, pp. 199–231, 08 2001.

[8] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of Machine Learning*. The MIT Press, 2012.

[9] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.

[10] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges," *computers & security*, vol. 28, no. 1, pp. 18–28, 2009.

[11] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Series in Statistics, Springer New York, 2013.

[12] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. New York, NY, USA: Cambridge University Press, 2008.

[13] W. Buntine, "Learning classification trees," *Statistics and Computing*, vol. 2, no. 2, pp. 63–73, 1992.

[14] Y. Mansour, "Pessimistic decision tree pruning based on tree size," in *Proceedings of the Fourteenth International Conference on Machine Learning*, pp. 195–201, Morgan Kaufmann, 1997.

[15] A. Liaw and M. Wiener, "Classification and regression by randomForest," *R news*, vol. 2, no. 3, pp. 18–22, 2002.

[16] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[17] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pp. 413–422, IEEE, 2008.

[18] F. E. Grubbs, "Procedures for detecting outlying observations in samples," *Technometrics*, vol. 11, no. 1, pp. 1–21, 1969.

[19] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, pp. 15:1–15:58, July 2009.

[20] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B*, vol. 39, no. 1, pp. 1–38, 1977.

[21] T. Krishnan and G. McLachlan, "The em algorithm and extensions," *Wiley*, vol. 1, no. 997, pp. 58–60, 1997.

[22] D. M. Blei, M. I. Jordan, *et al.*, "Variational inference for dirichlet process mixtures," *Bayesian analysis*, vol. 1, no. 1, pp. 121–143, 2006.

[23] "3GPP GTPv2." `https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=1692`, 2015. 3GPP Standard for the GPRS Tunneling Protocol.

[24] "3GPP GTPv2." `http://www.3gpp.org/about-3gpp/about-3gpp`, 2015. Info about the 3GPP organization.

[25] W. McKinney, "pandas: a foundational python library for data analysis and statistics,"

[26] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[27] C. J. van Rijsbergen, "Information retrieval," in *Information Retrieval 2nd edition*, 1979.