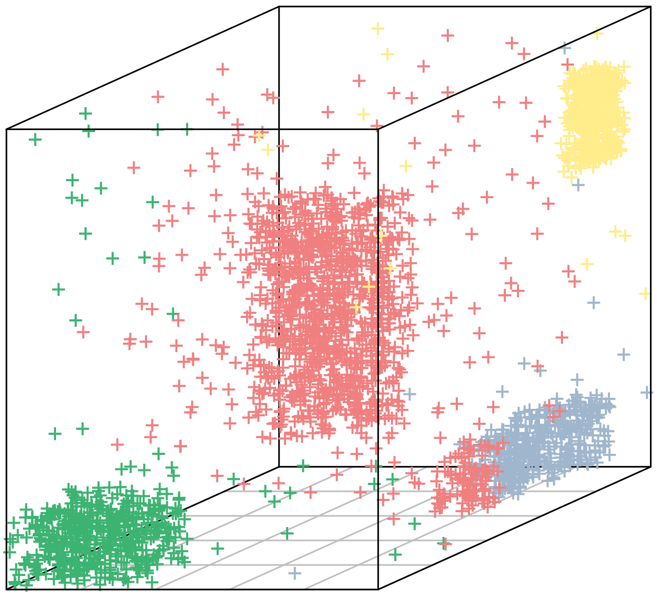
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für
Technische Informatik und
Kommunikationsnetze



Fabian Wanner

Anomaly Analysis using Host-behavior Clustering

Master's Thesis MA-2007-31
April 2007 to October 2007

Advisors: Daniela Brauckhoff, Arno Wagner
Supervisor: Prof. Dr. Bernhard Plattner

Abstract

In today's networks we are permanently confronted with huge amount of data. Monitoring such a network can not done by humans anymore. We need efficient and autonomous algorithms which investigate the data and probably raise an alarm if something abnormal is detected.

In this thesis we use a specific Data Mining technique called Clustering for building such a monitoring system. For each host a feature set gets calculated out of flow data attributes. This feature set is called profile and gives an exact description of the host behavior. All generated profiles are represented in a multi-dimensional feature space in which the clustering algorithm groups similar objects together.

There are three major steps to cope with. First we have to calculate host profiles out of NetFlow data. In a second step the most important features for clustering has to be determined by using a Feature Selection algorithm. The third step is the actual clustering process. In this step, we try to extract a behavior for every found cluster.

Our findings show, that the distribution of the features are heavy-tailed. That makes it hard for the cluster algorithm to find well separated clusters. We could not achieve a reliable monitoring system with our set up, which can detect a change in the behavior of the network and therefore is able to detect anomalies.

Zusammenfassung

In modernen Netzwerken ist man zunehmend mit einem grossen Datenaufkommen konfrontiert. Dieses lässt sich nicht mehr von Hand kontrollieren. Für eine umfassende Überwachung braucht es effiziente und autonome Algorithmen, die die Daten permanent kontrollieren und bei Ungereimtheiten Alarm auslösen.

In dieser Arbeit wird eine spezielle Data Mining Methode, genannt Clustering für den Aufbau eines solchen Überwachungssystems verwendet. Für jeden Rechner im Netz wird ein Satz von Attributen aus Flussdaten berechnet. Dieser Attributsatz wird Profil genannt und beschreibt das Verhalten des Rechners. Die Profile werden in einem multidimensionalen Raum dargestellt, in dem der Clustering Algorithmus die Rechner mit ähnlichem Verhalten gruppiert.

Die Arbeit ist in drei grosse Teile geteilt. Als erstes muss man aus Netzwerkverkehrsdaten Hostprofile erstellen. In einem zweiten Schritt müssen mit Hilfe eines Feature Selection Algorithmus die wichtigen Attribute herausgefiltert werden. Als letzter Schritt werden die Profile mittels des Clusterings gruppiert und es wird versucht, ein spezifisches Verhalten für jeden Cluster abzuleiten.

Die gewonnenen Resultate zeigen, dass die Verteilung der Attribute einer heav-tail Verteilung folgen. Das stellt den Clustering Algorithmus vor Probleme, da keine schön voneinander abgetrennten Cluster existieren. Daher war es nicht möglich, ein verlässliches Überwachungssystems aufzubauen, das die Erkennung von Anomalien zuverlässig zulässt.

Acknowledgments

The author would like to warmly thank the following persons: Daniela Brauckhoff, Arno Wagner and Bernhard Plattner for their help and advice; Alain Breedt and Thomas Steingruber for providing the technical equipment; Bernhard Tellenbach for its programming expertise; Maja Ott for correcting half of the report. . . unfortunately the other part was not written by then.

Contents

1 Introduction **1**
1.1 Problem Statement . . . 1
1.2 Report Overview . . . 2

2 Fundamentals **3**
2.1 Background . . . 3
2.1.1 Feature Subset Production . . . 3
2.1.2 Clustering . . . 4
2.1.3 Clustering of High-Dimensional Data . . . 7
2.1.4 Outlier Treatment . . . 7
2.2 Related Work . . . 8

3 Survey of Clustering Algorithms **11**
3.1 Incremental DBSCAN . . . 11
3.2 Fractal Clustering . . . 12
3.3 WaveCluster . . . 13
3.4 BIRCH . . . 13
3.5 GRIN . . . 14
3.6 O-Cluster . . . 16
3.7 Homogeneous Clustering . . . 16
3.8 Summary . . . 17

4 Design **21**
4.1 Building Host Profiles . . . 21
4.1.1 NetFlow . . . 21
4.1.2 Host Profiles . . . 22
4.2 Feature Selection . . . 24
4.2.1 SUD . . . 24
4.2.2 Correlation . . . 27
4.2.3 The Feature Selection Algorithm . . . 28
4.3 O-Cluster - A Closer Look . . . 28
4.3.1 The O-Cluster Algorithm . . . 29
4.3.2 O-Cluster Complexity . . . 31
4.3.3 The Sensitivity Parameter ρ . . . 31
4.3.4 Scoring of the Data Points . . . 32
4.4 Cluster Interpretation . . . 32
4.5 Profile Movement . . . 33

5 Implementation **35**
5.1 Building Host Profiles . . . 35
5.1.1 profcalc . . . 35
5.2 Feature Selection . . . 36
5.2.1 sud . . . 36
5.2.2 corr . . . 37
5.2.3 analyze . . . 38
5.3 Clustering . . . 38
5.3.1 bayes . . . 38
5.3.2 directclu . . . 39
5.3.3 clustat . . . 39
5.3.4 cluextr . . . 40
5.3.5 clucomp . . . 41
5.4 Profile Movement . . . 41
5.4.1 profmove . . . 41
5.5 Additional Tools . . . 42
5.5.1 normlize . . . 42
5.5.2 filter . . . 43
5.5.3 rmfeat . . . 43

6 Results **47**
6.1 Building Host Profiles . . . 47
6.1.1 Number of flows . . . 48
6.1.2 Calculation Speed . . . 48
6.2 Feature Selection . . . 50
6.2.1 Correlation . . . 50
6.2.2 SUD . . . 52
6.2.3 FS Algorithm . . . 57
6.3 Clustering . . . 58
6.3.1 The FS-Feature Set . . . 59
6.3.2 The whole Feature Set . . . 64
6.3.3 Parameter Settings . . . 65
6.3.4 Time Calculations . . . 66
6.4 Profile Movement . . . 66
6.5 Anomalies . . . 69
6.5.1 Malicious Host . . . 69
6.5.2 Blaster Worm . . . 73

7 Evaluation **81**
7.1 Profile Calculation . . . 81
7.2 Feature Selection . . . 81
7.3 Clustering . . . 82
7.3.1 Cluster Interpretation . . . 82
7.3.2 Cluster Evolution . . . 83
7.3.3 Clustering Algorithm . . . 83
7.3.4 Outliers . . . 84
7.3.5 Anomaly Detection . . . 84
7.3.6 Summary . . . 84
7.4 Profile Movement . . . 84
7.4.1 Single Host . . . 84

7.4.2 The whole Subnet 85
7.4.3 Summary 85

8 Conclusions **87**
8.1 Summary 87
8.2 Future Work 88
8.2.1 Profile Calculation 88
8.2.2 Feature Selection 88
8.2.3 Clustering 88
8.2.4 Profile Movement 88

A Raw Feature Table **91**

B Thesis Task **95**

List of Figures

2.1 Clustering Examples 4
2.2 Divisive Clustering Algorithm 5
2.3 Hierarchical Grid Structure 7

4.1 NetFlow Flow Extraction 22
4.2 Sliding Window Mechanism 23
4.3 Linked Hash List 24
4.4 Grid Cell Problem 29
4.5 O-Cluster Algorithm 30

5.1 Software Tool Set 45

6.1 Active Host Statistic 47
6.2 Average Flow Statistic (Normal Data) 48
6.3 Number of correlated Features 50
6.4 Number of correlated Features (Interval length) 53
6.5 Entropy Progress dependend on the Feature number 54
6.6 Heavy-tailed Distribution (15 Minutes) 55
6.7 Entropy Progress dependend on the Interval Length 56
6.8 Heavy-tailed Distribution (1 Minute) 57
6.9 Distribution of the Data Points (FS Set reduced) 60
6.10 Bayesian Cluster Solution (FS Set reduced) 61
6.11 Cluster Regions (FS Set reduced) 61
6.12 Direct Rule Cluster Solution (FS Set reduced) 62
6.13 Bayesian Classifier vs Direct Rule Classifier 62
6.14 Number of Clusters dependent on ρ 66
6.15 Number of Clusters dependent on the Number of Features 67
6.16 O-Cluster Performance 68
6.17 Average Profile Moving (one Day Interval) 68
6.18 Standard Deviation of the Profile Interval 69
6.19 Moving Behavior with Malware Infection 70
6.20 Malicios Host behavior vs. Average Host (Malware) 71
6.21 Moving Discrepancy from Average Value (Malware) 71
6.22 Moving Behavior by Botnet Participation 72
6.23 Malicios Host behavior vs. Average Host (Botnet) 73
6.24 Moving Discrepancy from Average Value (Botnet) 73
6.25 Flow Statistic during Blaster Outbreak 74
6.26 Number of correlated Features (Blaster) 75

6.27 Entropy progress (Blaster) 75
6.28 Number of Blaster infected ETH Hosts 76
6.29 Data Point distribution (Blaster) 77
6.30 Bayes vs. Direct Rule Classifier (Blaster) 78
6.31 Moving Behavior 78

Chapter 1

Introduction

Today we are increasingly confronted with huge amounts of data. Investigation of this data by humans is not practicable anymore. A good mechanism for analyzing this data automatically is data mining. Data mining tries to find rules and patterns in the data in order to gain information. We distinguish between two different types of data mining: Supervised Learning and Unsupervised Learning. In Supervised Learning we have for a given input vector a desired output vector (called test set) and we try to find a model which maps out this input-/output behavior as accurately as possible. After this learning step the model should be able to classify new input vectors correctly. In Unsupervised Learning we have no a priori output for a typical input vector. The algorithm tries to find some similarities between the given input vectors and group them together.

Input vectors consist of a set of different features. A feature or attribute is a part of the description of the considered element, for example the age can be a feature when we classify humans. Not all of these features are considered relevant for the learning algorithm. In order to extract only the relevant features, one often uses so called Feature Selection (**FS**) or Feature Extraction (**FE**) methods. Shrinking the feature set has the benefit of decreasing the dimensionality of the learning process and so reducing the complexity. It also makes the learning algorithm more efficient because it does not have to deal with probably misleading irrelevant features.

In this thesis we consider only one unsupervised learning method called clustering. Unsupervised learning is well suited for anomaly detection because no a priori knowledge of the attack classes is needed.

1.1 Problem Statement

The task of this Master Thesis is to find an optimal clustering mechanism for grouping hosts according to their traffic profiles, and to design and implement a system which realizes this optimal clustering mechanism in order to monitor and classify the behavior of hosts in the ETH campus network.

The goals of this thesis are:

Searching for Feature Selection and Clustering Mechanisms: In the first part, a search of existing mechanisms for feature selection and clustering has to be done. The most interesting algorithms are chosen for further evaluations. Furthermore, an induction into the subject matter should taken place.

Evaluation and Analysis of the Mechanisms: The most promising algorithms should be evaluated with respect to important requirements like time complexity, clustering quality, ability for processing high dimensional data and the like. The results should be summarised in an clustering survey. The selected clustering and feature selection algorithm should than be explained in more detail.

Implementation and Evaluation of a Monitoring System: In the last part, a monitoring system has to be implemented. It is important to take into account the huge amount of data and the possibility to run the system on off-the-shelf hardware.

Moreover, the system should provide an anomaly detection system which is able to detect dangers of every shape.

Deliverables: A practical and serviceable documentation of the work. The documentation should contain the clustering survey, a definition of the own approach to the problem, explanations about the implemented tool set and a final evaluation of the results.

1.2 Report Overview

The remainder of this thesis is organised in the following manner: In Chapter 2, a brief review of the most important Clustering and Feature Selection concepts is given. In the following Chapter, we propose and analyse a number of Clustering algorithms which are considered as the most promising for our purpose. Chapter 4 contains a description about the used algorithms and how they work in detail. In Chapter 5, a documentation about the used software tool set and its usage is given. In the next Chapter we present some experiments and their results. Followed by Chapter 7, an analysis and discussion of the found results is presented. In the terminal conclusions chapter, we present the final results and describe what topics we propose for future work.

Chapter 2

Fundamentals

In this Chapter a short overview about the different techniques in use like Feature Selection and Clustering is given. In the second part, previously done investigations and experiments are presented.

2.1 Background

2.1.1 Feature Subset Production

There is a big difference between Feature Selection and Feature Extraction. Feature Selection tries to find and select the most meaningful features among the available features. In contrast, Feature Extraction attempts to aggregate or combine the features in some way to extract the common information contained in them that is most useful for building the classification.

2.1.1.1 Feature Extraction

As stated above, Feature Extraction [1, 2] generates new features out of the existing ones. These new features are uncorrelated and retain as much variation as possible in the data set. One of the most common methods is the principal component analysis (**PCA**) [3]. The fact that PCA does not need any class labels for extracting the features, makes it well suited for Unsupervised Learning. With PCA we still need to store all the features, because they are needed to calculate the new features, so we do not save any memory. Another big problem with Feature Extraction arises in connection with comprehensibility. It will be difficult to get an intuitive understanding of the data when using these new extracted features. It will become very hard for humans to interpret the resulting clusters.

2.1.1.2 Feature Selection

A feature selection algorithm does not generate new features, it only searches for the most important features of the existing feature set. The algorithm tries to find a subset of features from the original set that ideally is necessary and sufficient to describe the whole model correctly. Many approaches exist for selecting these features. Primarily, we can divide these algorithms into two major categories: Wrapper models [4, 5, 6, 7, 8, 9] and Filter models [10, 11, 12, 13]. A wrapper chooses different subsets and makes a classification of the data using every subset separately. This classification is done by a clustering algorithm. At the end, all classifications are compared and the subset which composes the best classification is selected. Obviously this method is very time consuming. The filter algorithms choose a subset of features without applying clustering. They use other measurements like entropy to determine the importance of a feature. We can further divide unimportant features into irrelevant and redundant features. A feature is called irrelevant when it does not contain any important information for the clustering process. Redundant features carry information, which are already represented by other feature. Feature selection methods reduce the storage complexity.

2.1.2 Clustering

Clustering is an unsupervised learning method which tries to group objects with similar behavior (represented by the features) together. A similar behavior pattern between two objects is described by the similarity of each feature value between these data objects. Clustering is often confused with classification, but there is a certain difference between the two. In classification the objects are assigned to predefined classes, whereas in clustering the classes are yet to be defined.

In a two dimensional feature space the concept of clustering can be well illustrated. In Figure 2.1 we see three different data sets. We clearly can see

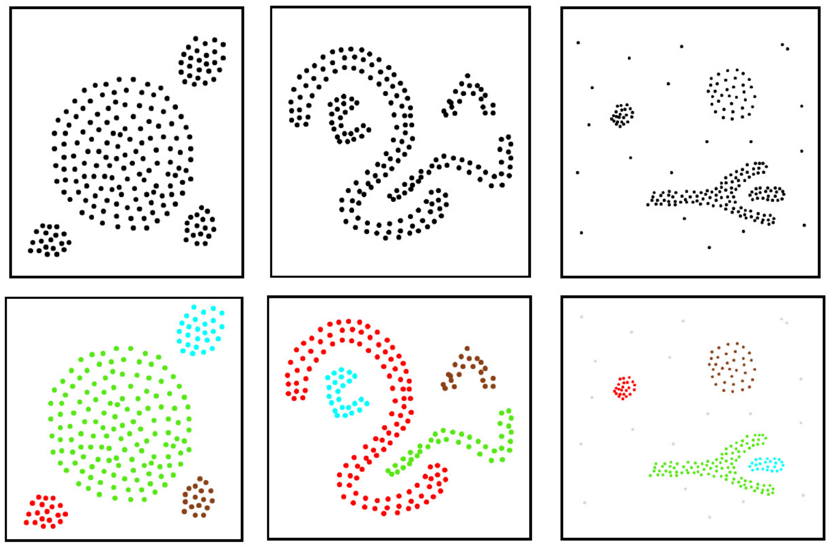


Figure 2.1: Examples of 2-dimensional Data Sets and a its possible clustering result (Figure adopted from Paper [14])

regions in which the data points are more concentrated than in others. Such groups describe objects which have similar behavior or are somehow similar in their characteristics. From now on these groups are termed as clusters. The resulting clusters in the three sample data sets are illustrated in Figure 2.1

Through this summarizing process, we can gain typical information about the objects in the clusters. Furthermore, we can now make statements for the clusters and for their representatives which could not have been made before.

There are many clustering methods available, and each of them may give a different grouping of a dataset. The choice of a particular method depends on the type of output desired, the known performance of method with particular types of data, and the size of the dataset. The most important methods are partitioning methods, hierarchical methods, density-based methods and grid-based methods.

2.1.2.1 Hierarchical Clustering

Hierarchical algorithms [15, 16, 17, 18, 19, 20] find successive clusters using previously established clusters from the last iteration. In the end, we have a cluster hierarchy in the shape of a tree. The hierarchical algorithms can be further divided into agglomerative (bottom-up) or divisive (top-down) methods.

Agglomerative: Agglomerative algorithms first assign each element to a separate cluster. In the next step they merge two clusters together, based on a determined measurement. When a certain condition is met, the algorithm stops. So they start from the bottom and evolve slowly to the top.

Divisive: Divisive algorithms first put all elements into one big cluster. Then they divide the cluster consequently into smaller clusters until a stop criterion is reached. This is where the name top-down results from.

There are generally two different stop criteria. If the distance between two clusters falls below a given threshold, or a defined number of cluster is found, the algorithm stops. The distance can be measured with the euclidean distance or another appropriate measurement.

Both types of hierarchical algorithms can be represented by dendograms. A dendogram is a tree diagram which illustrates the arrangement of the clusters. In Figure 2.2 an example of a divisive clustering algorithm is shown with its corresponding dendogram.

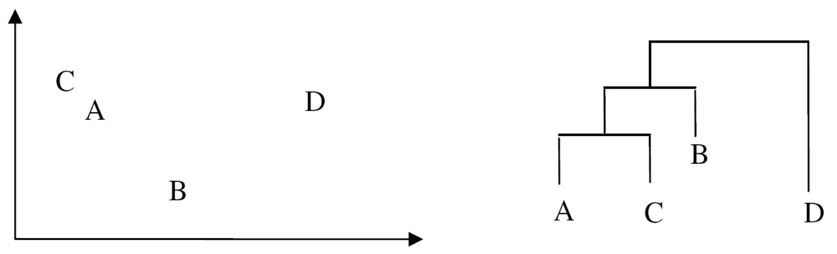


Figure 2.2: Examples of a divisive Clustering Algorithm with its Dendogram

2.1.2.2 Partitional Clustering

A partitional algorithm [21, 22, 23, 24, 25] determines all clusters at once. From the beginning we always have the same amount of clusters. In every iteration step the clustering gets more accurate. In partitional clustering, an element has to be in exactly one cluster and every cluster contains at least one element. The quality of the actual clusters is measured by using a quality function. In every iteration, the algorithm tries to maximize this quality function. Mostly, the quality function is based on a measurement of distance. The algorithm starts with an initial partitioning and then starts optimizing.

Partitional clustering can be divided into two major subcategories, the centroid and the medoid algorithms. In the centroid algorithms, the cluster center is represented by its center of gravity. The medoid algorithms represent each cluster center by means of the object which is closest to the gravity center.

2.1.2.3 Density-based Clustering

Density-based clustering algorithms [14, 26, 27] try to find clusters based on density of data points in a region. These types of clustering algorithms need a spatial feature space. An object belongs to a cluster if it has enough neighbors inside a determined radius or if it lies inside the radius of a cluster object. Otherwise, the density is too low and the objects are considered as noise.

A big advantage of density-based algorithm is their ability to find clusters with arbitrary shape.

2.1.2.4 Grid-based Clustering

Grid-based clustering algorithms [28, 29, 30]are related to density-based algorithms. They first quantify the clustering space into a finite number of cells and then perform the required operations on the quantified space. Cells that contain more than a certain number of points are considered as 'dense' and the dense cells are connected to form the clusters.

The grid can be formed into multiple resolutions by changing the size of the rectangular cells. Figure 2.3 represents a simple example of a hierarchical grid structure of three levels that is applied to a two dimensional feature space. In the case of d-dimensional space, hyper rectangles of d-dimensions correspond to the cells. In the hierarchical grid structure, the cell size in the grid can be decreased in order to achieve a more precise cell structure. As in Figure 2.3, the hierarchical structure can be divided into several levels of resolution. Each cell on the high level k is partitioned to form a number of cells at the next lower level $k + 1$. The cells at the level $k + 1$ are formed by splitting the cell on level k into smaller sub cells. In the case of Figure 2.3, each cell produces four sub cells on the next lower level.
Grid-based algorithms have some advantages in processing. Because they do not work on the actual dataset but on the grid cells. For this reason, it is possible to implement it by using parallel processing.

2.1.2.5 Summary

One of the main drawbacks in partitional clustering is the fact that we have to know the number of resulting clusters in advance. Especially with regard to the problem discussed in this thesis, the number of clusters can not be known. Often, partitional algorithms are simpler than hierarchical algorithms and so easier to implement. Hierarchical clustering algorithms are said to provide better clustering quality but have a higher time complexity as their partitional

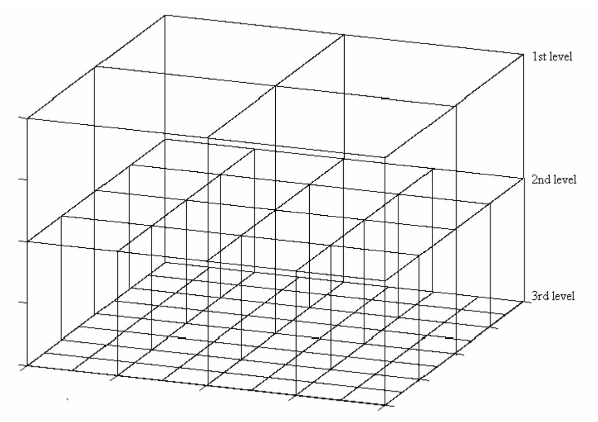


Figure 2.3: Examples of a hierarchical Grid Structure in a 2-D Feature Space

opponents. Density-based algorithms have the advantage that they can identify cluster with arbitrary shape, but the clustering quality is highly dependent on the chosen clustering parameters.

2.1.3 Clustering of High-Dimensional Data

Clustering high dimensional data sets involves some additional difficulties, and therefore is often called the curse of dimensionality. A data set can be considered as high dimensional if it has more than five dimensions.

Methods that rely on near or nearest neighbor information do not work well on high dimensional spaces. Especially when the space is sparsely filled, which is usually the case, this kind of measurement does not work anymore, because it is very unlikely that data points are nearer to each other than the average distance between data points. As a result, as the dimensionality of the space increases, the difference between the distance to the nearest and the most distant neighbors of a data object goes to zero. Distance based measurements are generally used in hierarchical clustering, partitional clustering and density-based clustering.

Grid-based clustering algorithms do not suffer from the nearest neighbor problem in high dimensional spaces. These kinds of algorithms face different problems when the dimensionality increases. The number of cells grows exponentially and finding adjacent high-density cells to form clusters becomes prohibitively expensive. If the data space is really sparse, the number of cells can be greater than the number of objects. This makes the calculation even more expensive.

2.1.4 Outlier Treatment

In most samplings of data, some data points will be further away from their expected values than what is deemed reasonable, so called outliers. This can be due to systematic error, faults in the theory that generated the expected values, or it can simply be the case that some observations happen to be a long way from the center of the data. Outlier points can therefore indicate faulty data, erroneous procedures, or areas where a certain theory might not be valid.

However, a small number of outliers is expected in normal distributions.

Outliers are eliminated by adapting their feature values to their environment. A possible method can for example be Clipping. In Clipping the extreme values are treated specially. The treatment can be divided into two different strategies:

Winsorizing: The extreme values are replaced by some other predefined values, for example by the mean value plus or minus three times of the standard deviation.

Trimming: In this strategy, the extreme values are just deleted and so ignored.

2.2 Related Work

Clustering techniques for anomaly detection have already been applied in previous research.

In the study of Wei et al [31], the authors use the agglomerative (bottom-up) clustering algorithm for grouping similar hosts together. Before clustering is applied, they filter the data and extract only the "active" hosts in order to minimize the data. In a second step, they build host profiles consisting of different features. The feature set is composed of direct and indirect features. The authors define direct features as the values which can be retrieved directly from the packet header and indirect features as values which are computed out of multiple packets in a host's communication session. A direct feature is for example an IP address or a port number, an indirect feature can be the average duration of a TCP connection. For the clustering process they choose the following feature subset: The number of distinct IP addresses contacted by the host, the total amount of bytes sent by the host, a list of open ports on the host and a measurement called communication similarity. This feature selection is done by empirical experience, no algorithm is used. The clustering is not done in real-time, they analyse pre-recorded data.

Xu et al [32] try to profile traffic in terms of communication patterns. The feature set used consists of source IP, destination IP, source port, destination port and the protocol field. In a first step, they extract significant clusters. In this process the protocol field is ignored, so the selected feature subset contains four values. It is not stated how they choose the features, but no Feature Selection or Feature Extraction method is applied. All of these four features are now considered separately. By using entropy, significant clusters are extracted along every dimension (feature). All object belonging to a cluster have exactly the same value for the considered feature, this feature is now called cluster key. For example, when I cluster along the source IP feature, all elements belonging to the same cluster have the same source IP. Because the elements in each cluster share the same cluster key, they only differ in the three remaining feature dimensions. In a second step, the authors build so-called behavior classes out of the existing clusters. In this behavior classes they group similar elements together. These behavior classes can be considered as "clusters in a cluster". This grouping is done very simply, they divide the three dimensional cluster box into 27 compartments (every axis is divided into three parts: $3 * 3 * 3 = 27$) and check in which compartment an element falls. As the name states, this behavior classes describe the behavior of a group of hosts. They apply their methodology to Internet traffic with the conclusion that three different typical behavior pattern of the clusters evolve. They differ in Server/Service Behavior (Web, DNS servers), in Heavy Hitter Host Behavior (NAT boxes) and in Scan/Exploit Behavior. They only analyse pre-recorded traffic and do not consider any real-time deployment. The method is time consuming, because they have to build a lot of clusters in every feature dimension.

The authors of [33] profile the hosts based on their transport-layer behavior. They use a graph-based structure to capture the interaction between the hosts. This graph structure is called a graphlet. As a starting point, they use the graphlets proposed by Karagiannis et al [34] and extend the concept further to meet their needs. The authors use the same feature set as the one in the BLINC paper. They do not apply any feature selection or feature extraction methods. The feature set is composed of the source IP, destination IP, source port and destination port. In a first step, they build an activity graphlet that captures all the current flow activity. This activity graphlet will continuously be updated. The second step compresses the large activity graphlet to a smaller, so called profile graphlet. This profile graphlet retains only the essential information from the profile graphlet. There will be one profile graphlet for every host which describes its behavior. Because of the fact that this graphlet evolves over time, they use an aging policy for dropping out obsolete and stale information. Otherwise, the graphlet would be overloaded and would not show the actual state of the host. In the experiments they updated the graphlets every fifteen minutes. This was possible because they captured traffic on an access link of a small company which has only 200 distinct internal IP addresses. The stated approach is meant to classify traffic at the edge of the Internet. Deploying this method on a backbone link is not practicable. Because of the big overhead with a graphlet for every node, the method will not scale very well. Furthermore, the paper presents no experiments which show how well this approach is suited for anomaly detection.

Unsupervised Niche Clustering for anomaly detection is proposed by the authors of [35]. This is a robust and unsupervised clustering algorithm that uses an evolutionary algorithm (**EA**) with a niching strategy. The authors use different data sets for testing their algorithm. One of these data sets consist of network traffic data. This data is made up of 42 different features, but they use only the numerical, non-zero features and so the set is reduced to 33 features. To reduce the feature set further, they apply the feature extraction method PCA. This leads to 21 new features which are used for clustering. Because the experiments are done with a pre-recorded data, it is difficult to say how well this approach is suited for real-time detection.

The paper from Burbeck et al [36] addresses real-time incremental clustering and is closely related to this thesis. For clustering, they use an incremental, hierarchical clustering algorithm called BIRCH [15]. A closer look into the BIRCH algorithm can be found in Section 3.4. Before anomalies can be detected, a clustering with only friendly data has to be done. Nevertheless, the method is still unsupervised. They do not know in advance what the clustering will look like. This learning phase is only for obtaining a clustering of a clean period in order to compare it with the actual period. This first clustering is the so called normality model. As stated above, BIRCH is an incremental clustering method and so every newly arrived data point can be included in the clustering iteratively. If the distance of the new data point to his cluster centroid is above a determined threshold, this data is considered as an anomaly and an alarm will be raised. The model can also learn new normal scenarios. When the normality drifts slowly enough, the cluster can adapt themself to the new situation automatically. Even if learning is not allowed, the model can easily be adapted to a new situation. Due to the incremental property, this new class of normal data can be incorporated into the model without the need to relearn the existing working normality model. The data set used in this work consist of 41 features, all of which are chosen for clustering, no feature selection or feature extraction method is used. This could be problematical, because irrelevant features can decrease clustering quality as is stated in [37]. Due to the linear time complexity of the BIRCH algorithm, this approach scales very well. This shows the huge amount of data they used for real-time clustering as well.

Summary

As we can see, most of the papers use almost the same four to five features. Most of these features are selected empirically, only paper [35] uses a feature selection algorithm. A good feature selection mechanism can probably improve the cluster quality and consequently improve the detection rate.

Chapter 3

Survey of Clustering Algorithms

Many different clustering algorithms were considered in this thesis. Because of the strong requirements stated, only a few algorithms ended up on a short list. These algorithms should be incremental, they should provide a clustering in only one scan through the data set (linear time complexity) and they should not be too sensitive, depending on the chosen parameters. The seven algorithms which are analysed more deeply are the density-based incremental DBSCAN [38], the grid-based algorithms WaveCluster [29] and Homogenous Clustering [39], the three hierarchical algorithms BIRCH [15], GRIN [16], and a mixture of different approaches called O-Cluster [40].

3.1 Incremental DBSCAN

Incremental DBSCAN is an extension of the normal DBSCAN [14] algorithm. DBSCAN is a density-based clustering algorithm. For that reason, the insertion and deletion of an object affects the current clustering only in the neighborhood of this object. The key idea of DSCAN is, that the density inside a cluster is higher than outside this cluster.

The algorithm needs two input parameters *Eps* and *MinPts* for building and expanding the clusters. The first parameter is the radius of the neighborhood of a given object. The second Parameter describes a density threshold. A cluster will now be expanded as long as the *Eps*-neighborhood of an object contains at least *MinPts* other objects. If the density falls beyond the threshold, the objects are considered as noise.

DBSCAN separate data objects into three classes:

Core object: These are objects that are at the interior of a cluster. An object is considered as a core object if there are enough other objects inside its *EPS*-neighborhood.

Border object: This is an object, that has not enough object ($< MinPts$) in its *EPS*-neighborhood to be a core object, but it falls within the neighborhood of a core object.

Noise objects: Any other object which is neither a core object nor a border object is a noise object.

To find a cluster, DBSCAN starts with an arbitrary instance in the data set and retrieves all instances of the data set with respect to *Eps* and *MinPts*.

Incremental DBSCAN yields to the same clustering results as the original DBSCAN algorithm but with the possibility of incrementally adding or deleting objects. For clustering n objects the time complexity is $O(n^2)$. With smarter data structures as for example R^*-Trees [41] are, the time complexity can be reduced to $O(n \log n)$. The R^*-Tree needs only $O(\log n)$ steps for calculating the *EPS*-neighborhood of an object instead of $O(n)$.

3.2 Fractal Clustering

Fractal Clustering is an incremental grid-based clustering algorithm. As the name states, the algorithm uses the so-called fractal dimension for clustering. A fractal is a geometric object with the property of self-similarity. Generally speaking, a small piece of the object looks exactly like the original object. Fractal sets are characterized by their fractal dimension.

The main idea behind FC is to group similar points into clusters in such a way that none of the newly included points change the cluster's fractal dimension radically. The data sets that we want to cluster are not required to consist of fractals, but rather that their clusters exhibit self-similarity over a range of scales.

The cluster algorithm is divided into two distinct steps:

Initialization Step: Before points can be added incrementally, we need an initial clustering. This is done in the initialization step. A procedure is applied to find a set of clusters, each cluster has to contain enough elements in order to compute the fractal dimension of the cluster. From the whole data set, an initial sample set is chosen at random. This initial set now gets clustered by an algorithm, this can be a traditional distance-based procedure.
The algorithm builds clusters by picking a random yet unclustered point and recursively finding the nearest neighbor in such a way that the distance between the point and the neighbor is less than a parameter κ. The neighbor is then included in the cluster, and the search for nearest neighbors continues in a depth-first fashion until no more points are left in the initial set.

Incremental Step: Now the feature space is divided into cells and each cell stores the amount of data points which fall into this cell. Every cluster consists of a set of cells. The remaining data is now incrementally added to the clustering. An object is included in the cluster, in which the inclusion has the smallest fractal impact. If the fractal impact exceeds a given threshold τ for all clusters, then the point is considered as noise.
It is possible that number and form of the cluster may change after having processed a set of data points. This can occur when the initial set was not a good representation of the distribution of the whole data set.

FC is a very fast algorithm, it generally has a time complexity of $O(n)$ for clustering a data set of n objects.

3.3 WaveCluster

The grid-based WaveCluster algorithm uses signal processing for clustering data. The multidimensional data-set is considered as a multidimensional signal. This signal is transformed into frequency space using wavelet transformation. In wavelet transformation, convolution with an appropriate kernel function results in a transformed space where the natural clusters in the data become more distinguishable. The clusters are then extracted by finding the dense regions in the transformed domain.

WaveCluster can broadly be divided into four parts:

Quantization: In the first step, the feature space has to be divided into cells. The number of this cells has an important impact on the performance of the clustering algorithm.

Transformation: In this step, discrete wavelet transformation will be applied on the quantized feature space. Applying this transformation to the cells, results in a new feature space and so in new cells as well. The resolution parameter r of the wavelet transformation determines the accuracy of the clustering.

Find connected components: This phase is needed for detecting the connected components. A connected component is composed of several cells and it is considered as a cluster.

Mapping objects to clusters: To every cluster and so to every cell the cluster number is assigned. The cells in the transformed space cannot be used directly to determine the clusters in the original feature space. The algorithm now does a mapping from transformed cells to the original cells using a lookup table. By using this lookup table, the clusters in the original feature space can be easily identified.

The time complexity of the quantization phase can be considered as a constant and in case where that the number of cells is much smaller than the number of elements in the data set, this time can be neglected. The general time complexity for WaveCluster is $O(n)$ for clustering n data objects.

3.4 BIRCH

The so called Clustering-Feature-Tree (CF-tree) on the basis of the Clustering Feature (CF) is the main part of the hierarchical BIRCH algorithm. This tree allows clusters to build incrementally out of multidimensional objects. A CF is a triple, summarizing the information about a cluster and its elements. When a new object arrives, the values of the related CF are adapted accordingly. A lot of objects can thus be taken together with only these three values. The nodes in the CF-tree consists of such CF Features. This makes the CF-tree to a compact representation of the whole data set. To build the CF-tree, the algorithm needs the branching factor value B and the threshold value T. The other parameter B describes how many successors a three node can have. The radius of each cluster has to be less than the threshold T. This threshold determines the accuracy of the representation of the data set and thus the size of the tree. The larger the tree, the more accurate the representation. In order to get the most exact representation, the tree should be as big as the free memory space.

The clustering process of BIRCH can be divided into four phases:

Phase 1: In this phase, the algorithm scans all data and builds an initial in-memory CF-tree using the given amount of memory. As each object is encountered, the CF tree is traversed, starting from the root and choosing the closest node at each level. When the closest "leaf" cluster for the current object is finally identified, a test is performed to see if adding the object to the candidate cluster will result in a new cluster with a radius greater than the given threshold T. This CF-tree tries to reflect the clustering information of the data set as accurately as possible. In this phase, possible outliers are eliminated. The time complexity of building the CF-tree out of n objects is $O(n)$.

Phase 2: This is an optional phase. When the resulting CF-tree is too big for an efficient clustering (see Phase 3), the tree will be further reduced.

Phase 3: This is the actual clustering process. In principal, all we have done with the CF-tree is a compression of our initial data set. In this phase we can apply any clustering algorithm, for example, k-means. The time complexity in this phase is highly dependent on the chosen clustering technique. By using k-means, we have a time complexity of $O(m^2)$, with m as the amount of CF elements in the CF-tree. This looks very bad at first sight, but we have to consider that $m \ll n$. After this step we have a clustering of our data.

Phase 4: The last phase is again optional and leads to cluster refining. The algorithm calculates the centroid of every cluster and checks every initial object, whether it belongs to the right cluster or whether it in fact belongs to a different cluster. As a result of doing so, the data needs to be scanned again. This phase has a time complexity of $O(n \times k)$, with n as the number of initial objects and k as the number of resulting clusters after Phase 3.

BIRCH has a big drawback, the clustering quality is highly dependent on the input order of the objects.

3.5 GRIN

GRIN is an incremental, hierarchical clustering algorithm based on gravity theory. One key idea behind the GRIN algorithm is that any arbitrarily shaped cluster can be represented by a set of spherical clusters. The GRIN algorithm assumes that all the incoming data objects are first buffered in an *incoming data pool*. The clustering algorithm can be divided into two phases.

Phase 1: The algorithm chooses at random a fixed amount of objects from the data pool. These objects are then clustered using the GRACE [42] algorithm, the gravity-based agglomerative hierarchical clustering algorithm. One big advantage of GRIN is the fact, that it is immune to the order of the incoming data instances. So the objects chosen at random do not affect the clustering quality. However, the order of the incoming data objects may impact the execution time of the second phase of the GRIN algorithm. After clustering through GRACE, we get a dendogram built out of these random samples. In a next step, the resulting clusters are tested, to see if they satisfy the *spherical shape condition.* Based on this test, the dendogram is flattened and pruned in order to derive the so-called *tentative dendogram.* The last step in this phase is optional and it would remove outliers from the dendogram and put them into a *tentative outlier buffer* for later use. The clusters in the tentative dendogram store only three values: The centroid of the cluster (geometric center), the cluster radius (maximum distance between the centroid and the objects) and the mass of the cluster (for example the amount of objects that the cluster contains).

Phase 2: The remaining objects in the input data pool are now examined one by one. Every data point is now checked as to whether it falls in the spheres of some leaf clusters in the tentative dendogram. For every such sample three different possibilities exist:

- The object falls into the sphere of exactly one cluster, then it is inserted into this leaf cluster.
- The object falls into the spheres of multiple clusters, then the gravity theory is applied to determine which leaf cluster the object belongs to. The cluster which performs the largest gravity force wins.
- In the last possibility, the object does not falls into any cluster. Then a test is conducted to check whether the object is currently an outlier or whether there is any cluster it could belong to without harming the spherical shape condition of this cluster. If the object is considered as an outlier it will be put into the tentative outlier buffer.

Once the tentative outlier buffer exceeds a threshold, the GRACE algorithm rebuilds the tentative dendogram using the objects in the outlier buffer and the leaf elements of the old tentative dendogram.

The GRACE algorithm has a time complexity of $O(n^2)$ for n objects. But GRACE clusters normally only a small subset of the original data set, so this quadratic time complexity is not so crucial. The insertion of m objects into the existing dendogram has a time complexity of $O(m)$. If the samples taken in the first phase are good representatives of the entire data set, then most of the objects in the second phase fall into one of the leaf clusters. In this case the time complexity of the GRIN algorithm is $O(n)$. But when the incoming data permanently forms new clusters, then the time complexity increases to $O(n^2)$.

3.6 O-Cluster

Orthogonal Partitioning Clustering (short O-Cluster) is a hybrid clustering approach consisting of grid-based, density-based and hierarchical methods. It is generally a further development of the OptiGrid clustering algorithm [43]. O-Cluster builds clusters like a hierarchical clustering algorithm using the divisive approach. The algorithm produces a binary cluster tree [1]. The splitting of the clusters is done until all data objects have been investigated or no significant improvement of the clusters is expected. The algorithm uses only objects which are in a fixed buffer of size *max_buffer*. If all objects fit into this buffer, the algorithm uses the whole data set. Otherwise, a technique called *active sampling* is used to determine which objects should be removed and which should be added.

The generation of clusters is done like a grid-based clustering algorithm. All clusters are represented by a grid structure. At the beginning all data objects are in one root cluster. For generating new clusters, a splitting criterion is used. This criterion works like a density-based method. The criterion searches for a valley in a given cluster. A valley is considered as a region of low density surrounded by regions with high density. If such a valley exists, the cluster gets divided along this valley and two new clusters emerge. In the tree structure, these two new clusters are the child nodes of the parent node. This procedure is repeated recursively. All these valleys are cutting planes, which build the grid structure of the space.

Apart from the buffer size, O-Cluster only needs one parameter, the *sensitivity* ρ $(0 \leq \rho \leq 1)$. This parameter determines the number of resulting clusters. A big ρ leads to more clusters, especially small clusters. A small value of ρ leads to a smaller number of clusters, because small clusters are filtered out.

The time complexity of O-Cluster is linear. For n objects and d dimensions we have a time complexity of $O(dn)$. The algorithm is well suited for real-time clustering.

3.7 Homogeneous Clustering

The incremental clustering algorithm proposed by Chen et al aims at generating summarised dendograms. The algorithm employs a statistic-based model to summarize the distribution of the similarity scores among the host profiles in the data set and to control formation of clusters. Due to the included summarization mechanism, the clustering algorithm produces highly concise dendograms for analysing the behavior. In order to do clustering, the algorithm requires six different parameters. All of them should be chosen empirically.

The clustering algorithm can be divided into the initialization phase and the incremental phase:

Initialization Phase: In this first phase, some randomly picked data points are extracted from the data set for constructing the initial dendogram. These data points are called the initial set. The initial prohibitively dendogram is built by using an ordinary agglomerative clustering algorithm

[1]Binary means that every nod has at most two child nodes.

such as single-link or complete-link.
A summarizing process is then performed on the initial dendogram to identify a set of representatives that provides an abstract description of the distribution of the initial set. Out of these representatives an abstract dendogram is constructed. This abstract dendogram serves as a basis for the incremental phase.

Incremental Phase: All the remaining data points from the data set are chosen one after the other and included in the clustering while the abstract dendogram is adapted accordingly to represent the changes. All objects which can not be easily included in the dendogram are stored in a temporary buffer for later use. As soon as the dendogram has to rebuild itself, these objects will be used, together with the representatives of each cluster.

The calculation of the time complexity is quite difficult. It depends on the chosen agglomerative algorithm in the first phase. In the case of single-link we have a time complexity of $O(n^2)$ or in the case of complete-link a time complexity of $O(n^2 \log n)$ for clustering n objects. The time complexity of the incremental process is more difficult to determine. The complexity is determined by how the amount m of so-called leaf homogeneous clusters increase when we permanently add new data points into the system. If m does not increase, we have a time complexity of $O(k)$ for inserting k objects. But in the worst case, we have a time complexity of $O(kn^2 \log n)$ for inserting k objects into a clustering already containing n objects.

3.8 Summary

We will now compare all of the considered algorithms. The most important qualities of the algorithms are the time complexity, the clustering quality, the algorithm parameters and the capability of clustering high dimensional data.

None of the algorithms has a quadratic time complexity for the average case. Only Homogeneous Clustering and GRIN can have a quadratic complexity in a worst case scenario. The most promising algorithms comparing the time complexity are BIRCH, Wave Cluster and Fractal Clustering, because they always have a linear time complexity.

With regards to the cluster quality, BIRCH suffers from two problems. The resulting clustering is highly dependent on the input order of the data points. The algorithm does not always produce the same clustering for an equal data set. The algorithm is said to be order-sensitive. The other problem with BIRCH is, that it may not work well when clusters are not 'spherical', because it uses the concept of radius or diameter to control the boundary of a cluster. All other algorithms can detect clusters of arbitrary shape.

The choice of the needed parameter is always a difficult task. BIRCH uses a global parameter for the cluster radius. This may lead to good results in some cases, but to bad results in other cases. On the other hand, the clustering quality of DBSCAN is highly dependent on the chosen parameters, too. The choice of the parameter is not independent of the distribution of the feature set. The Homogeneous clustering algorithm with its six parameters needs a lot more parameters than other algorithms. In order to get good clustering quality choosing all this parameters can be a difficult task. Good algorithms, considering the parameter setting, are GRIN and O-Cluster. Despite the nine parameters GRIN needs, the optimal parameter setting is not sensitive to the distribution of the data set. As our data changes with time, it is really important to have an algorithm which always delivers the same cluster quality without adapting the parameters. The O-Cluster algorithm needs only one parameter, which determines the sensitivity of the clustering. As the only algorithm, WaveCluster does not need an input parameter. But for transformation, an appropriate filter function has to be found.

Most of the algorithms are designed for clustering low dimensional data. WaveCluster and Fractal Clustering cannot handle high-dimensional data. Their performance decreases rapidly with the increasing size of dimensions. The time complexity for these algorithms is only determined with low dimensional data. BIRCH, GRIN, incremental DBSCAN and Homogeneous Clustering are also not practical in high dimensions, because their clustering approach relies on nearest neighbor information (see Section 2.1.3). On the other side, the author of the Homogeneous Clustering algorithm say that it is well suited for high-dimensional data.

The most promising algorithm for our purpose is O-Cluster. It has an acceptable time complexity, can detect clusters of arbitrary shape and it is well suited for high dimensional data. Furthermore, there is already an implementation of the algorithm available. In Table 3.1 a summary of the characteristic of each algorithm is given.

Algorithm	Type	High Dimensional	Time Complexity[a]	Number of Parameters
BIRCH	hierarchical	No	$O(n)$	2
GRIN	hierarchical	No	$O(n)$ WC $O(n^2)$	9
Homogeneous Clustering	hierarchical	Yes[b]	$O(n)$ WC: $O(n^2 \log n)$	6
incremental DBSCAN	density-based	No	$O(n \log n)$	2
WaveCluster	grid-based	No	$O(n)$	0 (Filter needed)
Fractal Clustering	grid-based	No	$O(n)$	1
OptiGrid / O-Cluster	grid-based	Yes	$O(dn)$ WC: $O(dn \log n)$	1

Table 3.1: Characteristics of the considered Clustering Algorithms

[a] n is the number of objects to cluster and d is the number of dimensions.
[b] According to the author

Chapter 4

Design

In order to investigate the behavior of a host, firstly we have to calculate a host profile for every host. We are currently considering only hosts belonging to the ETH subnet. The behavior of a host is determined with the aid of so-called features. A feature can, for example, be the number of outgoing TCP connections and the like.

Secondly, the number of features describing the host are minimized. This is done by using a Feature Selection algorithm. Minimizing the number of features is important because of performance reasons, the resulting system has to process the data in real-time and on of-the-shelf hardware.

The next step tries to group similar objects (hosts) together. This is done by applying a clustering algorithm. The resulting clustering can be considered as a summary of the behavior of all recorded hosts. This summary will then be used to detect and analyze anomalies.

4.1 Building Host Profiles

The data for building such host profiles is NetFlow v5 [44] data, which is collected on the SWITCH [45] border routers. SWITCH is the backbone provider of some public academic organizations like ETH. The NetFlow tool is invented by Cisco Systems [46] and its purpose is to provide a detailed view of the network behavior.

4.1.1 NetFlow

NetFlow is an embedded instrumentation within Cisco IOS Software to characterize network operation. IOS is the *Internetwork Operating System-Software*, it is the operating system of the Cisco routers and switches.

4.1.1.1 Flows

The NetFlow data is composed of so called flows. A flow describes a connection between two different hosts. NetFlow builds such flows by investigating IP packets. Each packet which passes the router is examined for a set of IP attributes, also called the packet fingerprint. This fingerprint determines, whether a packet belongs to an existing flow or whether it marks the beginning of a new flow. Figure 4.1 shows such a flow creation process.

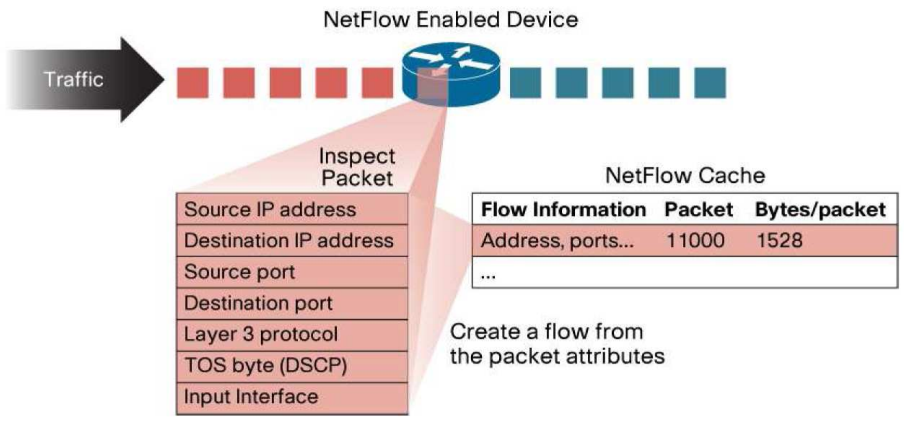


Figure 4.1: Flow Creation in a Router (Figure adopted from NetFlow paper [44])

The fingerprints consist of the following seven attributes: 1. IP source address, 2. IP destination address, 3. Source port, 4. Destination port, 5. Layer 3 protocol type, 6. Type of Service and 7. Router interface. If all attributes of two packets match each other, they belong to the same flow.

A flow will be exported and sent to the destination device (for example our anomaly detection system) when it is considered as finished. A flow can be considered as finished, when it is inactive for a long time or when the end is actually signaled. For example the FIN or RST flag is set in a TCP connection. If there is a long term connection between two hosts (for example an FTP download), then the connection can be divided into multiple flows.

NetFlow data is exported as a stream of UDP packets. Such packets consist of two parts, the NetFlow header and the NetFlow record. In the header part is general information about how many flows are included in the record, the NetFlow version and so on. In the record part is the actual information about the individual flows.

4.1.2 Host Profiles

In this step, host profiles are calculated out of NetFlow flows. A host profile consists of 41 different features. These features can be found in Table A.1. The idea in this phase is to calculate as much features as possible without thinking about how meaningful they could be. The task about identifying important features will be done by the Feature Selection algorithm. All host profiles generated for each active host in a typical time interval are now referred to as profile interval.

These 41 features of each host profile are calculated from the attributes delivered by the flows. A flow consists of several attributes; the following are taken when calculating the host profiles:

addr: The IP address of the connection initiator.

dstaddr: The IP address of the connection receiver.

dPkts: The Number of packets in this flow.

dOctets: Number of Layer 3 bytes in this flow.

First: Time stamp of the beginning of the flow.

Last: Time stamp at the end of the flow.

port: TCS/UDP source port number or equivalent.

dstport: TCP/UDP destination port number or equivalent.

prot: IP protocol number.

These host profiles are built over different time intervals. From five minutes up to one day, in order to determine how precise just a quick look on the net of a few minutes is, compared with a look lasting a whole day.

Additionally, a sliding window mechanism for calculating the profiles can be used. With this mechanism, we can achieve a smaller change of the profiles in consecutive time intervals. For every new interval, the Windows are slided sideways by one. The technique used is illustrated in Figure 4.2.

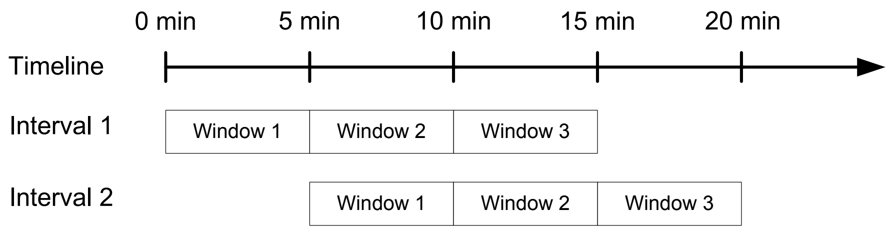


Figure 4.2: Sliding Window Mechanism with three Windows (Window size: 5 min)

4.1.2.1 Calculation Details

For the calculation of designated features, we have to store additional information during the interval duration. For example the calculation of the feature *num_diff_dest_ports* for a specific active host. All ports which appear in a flow belonging to this host have to be stored. The value is first looked up in the data base and then possibly stored. This look up and storing procedure has to be fast in order to achieve the time constraints.

For an efficient storing and look up process we used a simple singly-linked list combined with a hash function, called *simple_hash_list*. The hash function needs as parameter the hash value h. This value determines the number of different linked lists for each feature[1]. For this reason we do have a two dimensional linked list. The organisation of the list can be seen in Figure 4.3.

[1]Not each feature needs always a separate list, we can sometimes calculate different features out of the same two dimensional list.

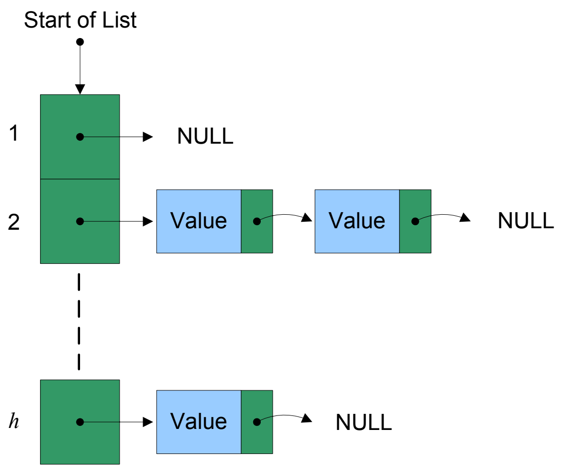


Figure 4.3: Linked Hash List

At the beginning, a pointer array of list pointers with size h is allocated. We have now h different list slots where objects can be inserted. The slot number s where the object with value v gets inserted is determined by using the following hash function:

$$s = v \bmod h \tag{4.1}$$

The advantage of this hash function is, that the algorithm needs not to check the whole construct for an element. The algorithm can only check the belonging list slot. It is obvious, the bigger the hash value, the faster the look up and storing process. In order to get a good distribution of the objects among the list slots, it is recommended to take a prime number for the value of h.

4.2 Feature Selection

Feature Selection is done by using two different Algorithms. For one thing it is the SUD [47] algorithm and for another it is the pairwise correlation between all features. Based on the result of both algorithms, the most relevant features are chosen for clustering.

4.2.1 SUD

The SUD algorithm addresses the problem of selecting a subset of features for clustering from a data set of unsupervised data, that is to say data without class information. The core part of SUD is an entropy measure that determines the important original features in the data set. Unlike other unsupervised feature selection algorithms, SUD never applies clustering for selecting these important features. SUD is a pure filter algorithm.

The key idea of SUD is the fact that removing an irrelevant feature from the set of attributes may not change the underlying concept of data, but not so vice versa. So we want to remove as many variables as possible but still maintain the level of distinctness as if no variables had been removed. Let us consider an instance in a data set which is described by a set of features. If the data has distinct clusters, this instance should belong to at least one cluster. There should also be some instances which are very close to the considered one and so belong to the same cluster, and some which are well separated from this instance and belong to some other clusters.

4.2.1.1 The Entropy Measure

The entropy measure in SUD does not need class information to evaluate the features unlike some other entropy measures like ID3. From the definition of entropy we know, if the probability is uniformly distributed we are most uncertain about the outcome, and so the entropy is maximum. This will happen when the data points are uniformly distributed in the feature space. On the other hand, when the data has well-formed clusters, the uncertainty is low as is also the entropy. Generally speaking, the entropy is a measure of uncertainty.

The entropy measure E used here is based on a similarity measure S between two instances of the data set. This similarity measurement is again dependent on the Distance D between these two instances. If these two instances are very close to each other the similarity is high, if they are very far apart the similarity is small. When the instances are either really close or really far apart, we know they belong either to the same cluster or to different ones. In both cases the entropy should be small. But when the two instances are neither far nor close, we cannot say to which cluster they belong and so the entropy should be high.

An instance is denoted as x_i, where $i = 1 \dots N$. So we have a data set containing N instances, and every instance consists of M features. x_{ik} would be the k^{th} feature value of the i^{th} data point.

The used distance measure is the Euclidean distance. The distance D_{ij} between two instances x_i and x_j is defined as follows:

$$D_{ij} = \sqrt{\sum_{k=1}^{M} \Big(\frac{x_{ik} - x_{jk}}{max_k - min_k} \Big)^2} \tag{4.2}$$

Where max_k is the biggest value of feature k over all instances and min_k is the smallest value of feature k over all instances. The expression $(max_k - min_k)$ causes a normalization of the distance.

With the use of this distance, the similarity S_{ij} between this two instances is calculated:

$$S_{ij} = \mathrm{e}^{-\alpha \cdot D_{ij}} \tag{4.3}$$

The parameter α determines the curvature of the e function. Because the distribution of the features is not known in advance, an appropriate α has to be calculated. As a result of the exponential function, the similarity S can lie between 0 and 1. The entropy is at maximum at similarity 0 as well as at similarity 1. And so the entropy is minimal at similarity 0.5. This minimal entropy is reached by data points which are neither close nor far away from each other.

The authors of the SUD paper consider the mean distance between all data points as a good distance for describing this case. And so α can be calculated by assigning the value 0.5 to the average similarity $\overline{S}$:

$$\begin{aligned} \overline{S} &= 0.5 = \mathrm{e}^{-\alpha \cdot \overline{D}} \\ \Rightarrow \alpha &= \frac{-\ln 0.5}{\overline{D}} \end{aligned}$$

With $\overline{D}$ as the average distance between all instances. The entropy E of the data set is finally defined as:

$$E = -\sum_{i=1}^{N}\sum_{j=1}^{N}\Big(S_{ij} \cdot \log_2 S_{ij} + (1 - S_{ij}) \cdot \log_2(1 - S_{ij})\Big) \tag{4.4}$$

4.2.1.2 The Algorithm

We use the following Sequential Backward Selection algorithm to determine the relative importance of each feature. The pseudo code is taken over from the SUD paper [47]:

SUD-Algorithm

```
SUD(N){
    $T$ = Original Feature Set
    for k = 1 to M-1{ /* Iteratively remove features one at a time */
        for every variable v in $T${ /* Determine which feature to remove */
            $T_v$ = $T$ - v
            Calculate $E(T_v)$ on N
        }
        Let $v_k$ be the feature that minimizes $E(T_v)$
        $T$ = $T$ - $v_k$ /* Remove $v_k$ as the least important feature */
        Output($v_k$)
    }
}
```

in each iteration step, the entropy E is calculated after removing one feature from the set of remaining features. A feature is removed as the least important variable if the set without the variable gives the least entropy. The removal of this feature reduces the uncertainty the most. This procedure is continued till the importance of all variables is determined.

A difficult task is to determine the border at which we should stop with the algorithm and take all the remaining features. Because the entropy will decrease all the time and reach the smallest value when there is only one feature left.

4.2.1.3 Complexity

The SUD algorithm is very time costly. The calculation of the entropy has quadratic complexity with the number of data instances N. But this entropy has to be calculated several times. During the calculation of the entropy, the number of features decreases permanently, and so the complexity of the distance measurement ($O(M)$) decreases as well. But this complexity is linear and can be neglected because of the fact that there are many more data instances N than features M ($N \gg M$).

Considering the SUD Algorithm in Section 4.2.1.2, we can see that the entropy has to be calculated several times. In the first step one feature is always removed and the entropy is calculated without it. These are already M calculations of the entropy. In the next step one feature has disappeared because it has been dropped by the algorithm. Now the entropy has to be calculated $M-1$ times. This continues until only one feature remains. The number of entropy calculations can be determined as follows:

$$M + (M-1) + \ldots + (M-M+2) = \sum_{2}^{M} x = \frac{M(M+1)}{2} - 1$$

The minus one on the extreme right of the expression above, comes from the fact, that the entropy does not need to be calculated when only one feature remains. Out of this formula, the total complexity of the SUD algorithm can be calculated as follows:

$$\begin{aligned} O\left(N^2 \cdot \left(\frac{M(M+1)}{2} - 1\right)\right) &= O\left(N^2 \cdot \left(\frac{M^2}{2} + \frac{M}{2} + 1\right)\right) \\ &\approx O(N^2 M^2) \end{aligned} \tag{4.5}$$

The time complexity of the SUD algorithm is therefore quadratic in the number of data instances N and in the number of features M.

4.2.2 Correlation

Another important measurement is the correlation. It indicates the strength and direction of a linear relationship between two features. The correlation can take values between -1 and 1. The closer the coefficient is to either -1 or 1, the stronger the correlation between the variables. A value of 1 (and -1 respectively) means, that two variables are completely positive (and negative respectively) linearly dependent. If the value is 0, then the two features are not linearly dependent. The correlation coefficient detects only linear dependencies between two variables, so two features with correlation 0 can still be dependent in a non-linear sense.

To do Feature Selection with correlation, we have to calculate the correlation matrix. In this matrix every feature gets correlated with every other. As a summary of all correlations of a feature, the mean correlation will be computed. The smaller this correlation is, the more independent is the feature. When we have a high correlation, the feature is unimportant, because its information can be represented by other features.

4.2.2.1 Correlation Coefficient

Each of the N data instances has M different features. The correlation coefficient between two features x and y can be calculated as follows:

$$r_{xy} = \frac{\frac{1}{1-N}\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\frac{1}{1-N}\sum_{i=1}^{N}(x_i - \bar{x})^2}\sqrt{\frac{1}{1-N}\sum_{i=1}^{N}(y_i - \bar{y})^2}} \tag{4.6}$$

$\bar{x}$ and $\bar{y}$ are the expectation values. They can be computed like this:

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i \qquad \bar{y} = \frac{1}{N}\sum_{i=1}^{N} y_i \tag{4.7}$$

4.2.3 The Feature Selection Algorithm

In order to specify the relevant features we combine the two techniques SUD and Correlation. The Correlation is used to determine the irrelevant features and then they are dropped according to their importance as calculated by SUD. The Algorithm needs a correlation threshold T as input. This threshold specifies the border, at which a feature is considered as correlated. The exact FS-Algorithm looks as follows:

FS-Algorithm

```
FS(N, T){
    $F$ = Original Feature Set
    while($r_{xy} > T$){ /* Is there a correlation bigger than the threshold */
        search biggest $r_{xy}$ /* Choose features $x$ and $y$ with the highest corr */
        if($\text{rank}_{SUD}(x) > \text{rank}_{SUD}(y)$){
            $F = F - x$ /* Drop Feature $x$ */
        }
        else{
            $F = F - y$ /* Drop Feature $y$ */
        }
    }
    while($F > 0$){
        search biggest $\text{rank}_{SUD}(x)$
        add_to_ranking($x$)
        $F = F - x$
    }
}
```

The FS-Algorithm produces a ranked List with the most important features. The algorithm does not deliver a specific number of features which should be used for clustering. It only shows the importance of the features to the user. The user still has to choose the number of features he wants to use for clustering himself.

4.3 O-Cluster - A Closer Look

In this section we want to investigate the O-Cluster algorithm in a deeper way, because this algorithm is very well suited for our purpose, as we have stated in Chapter 3.

O-Cluster uses a grid-based approach for clustering, but as we have mentioned in Section 2.1.3 Grid based algorithms have serious problems with high dimensional data sets. It is possible that clusters are split by some of the (d-1) dimensional cutting planes and the data points of the cluster are spread over many grid cells. By cutting the space into cells, the natural neighborhood between the data points gets lost. In a worst case scenario, such clusters cannot be identified any more. Figure 4.4 shows such a situation in a three dimensional case. Almost every object falls into a different grid cell and the cluster cannot be identified any more. In a high dimensional space, such a situation is much

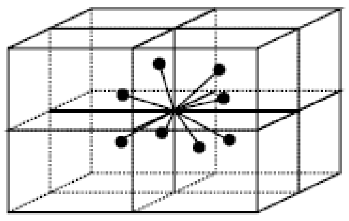


Figure 4.4: Worst case scenario (Figure adopted from OptiGrid paper [43])

more likely to occur than in low dimensional spaces.

For this reason, O-Cluster uses a different technique for building the grid. This technique avoids the problem by determining cutting planes which do not partition clusters. The algorithm constructs a grid-partitioning of the data by calculating the partitioning hyperplanes using contracting projections of the data. It looks for hyperplanes that satisfy two requirements:

- The separating hyperplanes should cut through regions of low density relative to the surrounding regions.
- A cutting plane should discriminate the clusters as much as possible.

The first requirement makes sure that a cutting plane does not split a cluster. The second constraint is important for detecting the clusters. Without this constraint, the algorithm would find the best cutting planes at the borders of the space, because of the minimal density there. O-Cluster uses a statistical test to validate the quality of the cutting plane.

The generated cutting planes are always axis-parallel. The authors of [43] show that the error introduced by axis-parallel partitioning decreases exponentially with the number of dimensions in the data space. This validates the use of axis-parallel projections as an effective approach for separating clusters in high dimensional spaces.

4.3.1 The O-Cluster Algorithm

O-Cluster is a recursive algorithm. The algorithm evaluates possible splitting points for all dimensions in a partition, selects the 'best' one, and splits the data into two new partitions. The outline of the clustering algorithm can be found in Figure 4.5.

1. **Load buffer**: If not all data points fit into the buffer, a random sample is used. The algorithm uses all points in the initial buffer as a root partition.

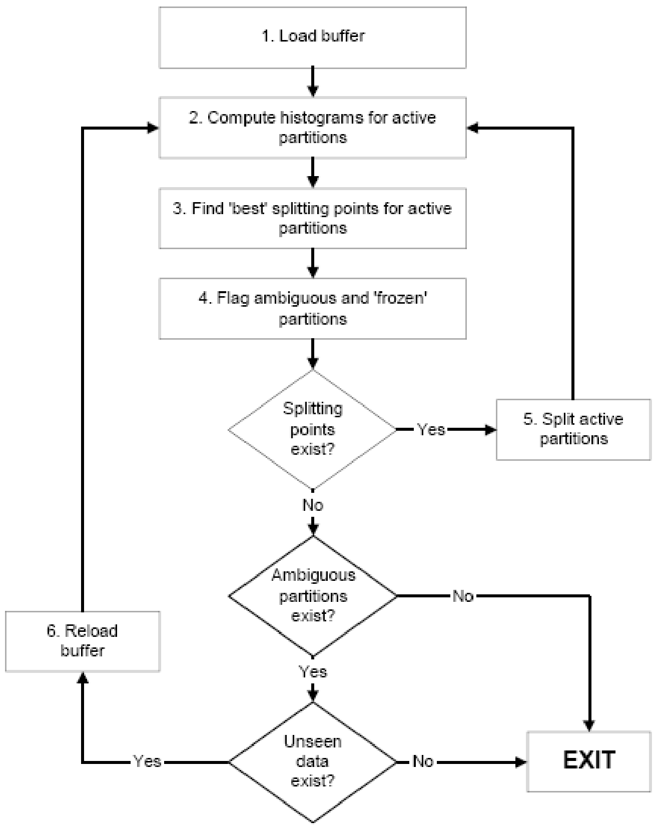

1. Load buffer
2. Compute histograms for active partitions
3. Find 'best' splitting points for active partitions
4. Flag ambiguous and 'frozen' partitions
Splitting points exist?
Yes
5. Split active partitions
No
Ambiguous partitions exist?
No
Yes
6. Reload buffer
Unseen data exist?
Yes
No
EXIT

Figure 4.5: O-Cluster Algorithm (Figure adopted from O-Cluster paper [40])

2. **Compute histograms for active partitions**: Any partition not marked explicitly as ambiguous or 'frozen' is considered active. An explanation about ambiguous or 'frozen' partitions is given in Step 4. For all active partitions the algorithm determines a set of projections and computes histograms along these projections. It is essential to compute histograms that provide good resolution but also that have data artifacts smoothed out.

3. **Find 'best' splitting points for active partitions** O-Cluster uses the generated histograms to find the 'best' valid cutting plane. A valid cutting plane passes through a point of low density (a valley) in the histogram. Additionally, the point of low density should be surrounded on both sides by points of high density (peaks). O-Cluster attempts to find a pair of peaks with a valley between them where the difference between the peak and the valley histogram counts is statistically significant. Statistical significance is tested using a standard χ^2 test:

$$\chi^2 = \frac{2(observed - expected)^2}{expected} \geq \chi^2_{\alpha,1}$$

The observed value is equal to the histogram count of the valley and the expected value is the average of the histogram counts of the valley and the lower peak. The implementation in the Oracle Data Miner [48] uses a 95% confidence level ($\chi^2_{0.05,1} = 3.843$). Since multiple splitting points can be found to be valid separators per partition according to this test, O-Cluster chooses the one where the valley has the lowest histogram count as the 'best' splitting point. Thus the cutting plane would go through the area with lowest density.

4. **Flag ambiguous and 'frozen' partitions**: If no valid splitting point is found in the actual partition, O-Cluster checks whether the χ^2 test would have found a valid splitting point at a lower confidence level (e.g., 90% with $\chi^2_{0.1,1} = 2.706$). If that is the case, the current partition can be considered ambiguous, because more data points are needed to establish the quality of the splitting point and to make a final decision. If no splitting points are found and there is no ambiguity, the partition can be marked as 'frozen' and the records associated with it marked for deletion from the active buffer. This partition cannot be further divided into smaller clusters.

5. **Split active partitions**: If a valid splitting point exists, the data points are split into two new active partitions along the cutting plane. For each new partition the processing proceeds recursively from Step 2.

6. **Reload buffer**: If all existing partitions are marked as 'frozen' and there are no more data points available, the algorithm exits. Otherwise, it reloads additional unseen data records into the buffer for the ambiguous partitions. The new data points will replace the records belonging to 'frozen' partitions. Once the buffer reload is completed, the algorithm proceeds from Step 2. The algorithm requires, at most, a single pass through the entire data set.

4.3.2 O-Cluster Complexity

The histogram computation step is of time complexity $O(Nd)$ where N is the number of data points in the buffer and d is the number of dimensions. The selection of the best splitting points for a single dimension is $O(b)$ where b is the average number of histogram bins in a partition. Choosing the best splitting point over all dimensions is $O(db)$. The assignment of data points to newly created partitions requires a comparison of an attribute value to the splitting point and the time complexity has an upper bound of $O(N)$. Loading new records into the data buffer requires their insertion into the relevant partitions. The time complexity associated with scoring a record depends on the depth of the binary clustering tree s. The upper limit for filling the whole active buffer is $O(Ns)$. The depth of the tree depends on the data set.

In general, the total time complexity can be approximated as $O(Nd)$.

4.3.3 The Sensitivity Parameter ρ

The effect of creating spurious clusters due to splitting artifacts can be alleviated by using O-Cluster's sensitivity parameter ρ. ρ is a parameter in the [0, 1] range that is inversely proportional to the minimum count required to find a histogram peak. A value of 0 requires the histogram peaks to surpass the count corresponding to a global uniform level per dimension. The global uniform level is defined as the average histogram count that would have been observed if the data points in the buffer were drawn from a uniform distribution. A value of 0.5 sets the minimum histogram count for a peak to 50% of the global uniform level. A value of 1 removes the restrictions on peak histogram counts and the splitting point identification relies solely on the χ^2 test.

4.3.4 Scoring of the Data Points

Because there is an implementation of the algorithm available, we will not implement it by ourself. We will use the Oracle Data Miner (ODM), a software developed by ORACLE [48]. This software has some additional specialties by assigning the cluster identifier to the corresponding data points.

4.3.4.1 Bayesian Classifier

After clustering with the O-Cluster algorithm, the ODM delivers a set of rules. These rules determines which data object belongs to which cluster. In order to get the cluster ID for each object, ODM has to apply the clustering model to the actual data. ODM does this by using a Bayesian classifier.

Oracle describes the scoring as follows: “The clusters discovered by O-Cluster are used to generate a Bayesian probability model that is then used during scoring (model apply) for assigning data points to clusters. The generated probability model is a mixture model where the mixture components are represented by a product of independent normal distributions for numerical attributes and multinomial distributions for categorical attributes.” Unfortunately, there is no exact description about the algorithm and how the scoring is done.

The reason why ODM does not directly apply the found rules to the data objects is because of the possible sampling. There could be some data points which have not participated in the model building process. For this reason, a classifier is needed.

4.3.4.2 Direct Rule Classifier

By choosing a big input buffer, all data points are used for building the clustering model. In this case, no probability based classifier is needed. The data points can directly be assigned to the clusters. This sort of classification is now called Direct Rule Classifier. The assignment of the data points to the relevant cluster is done by using the cluster rules generated by ODM.

4.4 Cluster Interpretation

After having clustered the active profiles, it may be possible to assign a typical behavior to each cluster. For example, a cluster containing web servers or a cluster only containing VoIP [2] devices. Two different approaches are used to determine the actual cluster behavior:

[2]**V**oice over **I**nternet **P**rotocol

Data aspect: The cluster behavior can be described by searching some well known hosts. For example the ETH DNS servers or the mail servers. Also the hosts closest to the cluster center are being searched and investigated according to their behavior.

Feature aspect: The interpretation of the features can lead to the cluster behavior.

Both approaches may have some drawbacks. Clusters can be arbitrary shaped, and so the cluster center can probably lie outside of the actual cluster region. This can cause some problems for the data approach. In this case the host which is nearest to the cluster center can give no information about the behavior of the cluster. Furthermore, it can even give some misleading information about the behavior. Another problem by using the data approach can occur, when no well known host is inside the cluster. Investigating the function of an arbitrary host is a difficult task.

The feature aspect approach can be problematical when the actual features are used for clustering. Some of them are very hard to interpret and thus to determine a cluster behavior.

4.5 Profile Movement

Another method for analysing the host behavior is the distance the host moves in the feature space. This distance is now called thee moving distance. In order to achieve practical results, these distances should be calculated between two consecutive time intervals. For example, the distance moved by a host between two intervals which is several times bigger than its average movement could point to a possible anomaly.

Chapter 5

Implementation

The software developed for this thesis is organised in modules. For every specific task, there is a separate software tool. The used tool set and its data flow is depicted in Figure 5.1. The files generated by the tools lie always in the working directory. This is the directory where the user has started the tool from. Nevertheless, the tool has not to lie in this directory.

5.1 Building Host Profiles

5.1.1 profcalc

The module for calculating the host profiles out of the NetFlow data is called **profcalc**. It reads both, bzip2-compressed and uncompressed NetFlow .dat files and generates profile intervals out of it. There are several parameters which can be passed to the tool:

-c <value>	With this option, the type of the generated profile interval can be determined. A value of 0 produces an ARFF[1] file, which is a often used file format by data mining tools. Passing a value of 1 generates profiles, which can be investigated with the Hierarchical Clustering Explorer [49]. And finally, a value of 2 produces files for the O-Cluster algorithm, which can be processed by the ODM. The default value is 0.
-d <directory path>	With this option we can specify the path to the NetFlow data. The folder can contain either only '19993' and '19991' files or both. If the folder contains both file types, the resulting profiles are composed.

[1]Attribute-Relation File Format, it is developed by the Machine Learning Project at the Department of Computer Science of The University of Waikato.

-i <**interval**>	This parameter specifies the profile interval length. It is in the format *days:hours:minutes*. For selecting a fifteen minutes interval we have pass 0:0:15 on the command line.
-l	Prints a list of the calculated features.
-m <**value**>	This value determines the number of memory slots for the *simple_hash_list* (See Section 4.1.2.1). The more memory slots the structure has, the faster a searched object can be found.
-o <value>	Determines the offset *o* in the directory. The first taken NetFlow file will be the *o*-th one counting from the beginning.
-p <**period**>	With this option we can specify the time period over which the intervals are calculated. It has the same format as the interval parameter **-i**. It should be at least as big as the interval length.
-s <**subnet**>	With this parameter we can determine the subnet we want to build the profiles for. The default is the ETH subnet: 129.132.0.0/16
-w <**value**>	This value determines the number of windows the interval **i** gets divided into.
-h	Prints a usage message.

In the end we get some consecutive numerated files with the starting time of the interval inside the file name. The file name starts always with *FileX...*, the *X* stands for the file number. The files are consecutive numerated starting from 1. If no directory is specified, we can pass the input file as the last argument. In this case only this single file gets processed.

5.2 Feature Selection

5.2.1 sud

The **sud** tool calculates the SUD ranking for each profile interval and saves a statistic into a file. We can pass the following arguments on the command line:

-a	Makes the tool analysing ARFF Files.
-c <**value**>	Determines the number of CPU cores which should be used for calculating the SUD ranking. By using this parameter, the calculation becomes parallel and all much faster.
-d <**directory path**>	With this option we can specify the path to the data files.

-f	Makes the execution twice as fast, but less accurate. In Formula 4.4, we see the entropy between every two data points is calculated twice. The new formula can be seen in Formula 5.1.
-n <value>	This value determines the number of files which should be taken from the directory. Maybe the directory contains 100 files but we just want to take 10.
-O	Makes the tool analysing O-Cluster files.
-o <value>	Determines the offset *o* in the directory. The first taken NetFlow file will be the *o*-th one counting from the beginning.
-p <value>	With this parameter we can determine the percentage of the data set we want to take for feature selection.
-s <value>	This seed is only needed with the **-p** option. The Seed determines which elements are randomly picked. Default value of the seed is the actual time of day. This means multiple runs without specifying a seed wont lead to similar results. To achieve this, always the same seed has to be passed to the tool.
-h	Prints a usage message.

The resulting files have the same file name as the analysed one, but with an additional *sud* substring in the name. If no directory is specified, we can pass the input file as the last argument. In this case only this single file gets processed. By using the **-f** option, the entropy gets calculated as follows:

$$E = -\sum_{i=1}^{N}\sum_{j=i}^{N}\Big(S_{ij} \cdot \log_2 S_{ij} + (1 - S_{ij}) \cdot \log_2 1 - S_{ij})\Big) \tag{5.1}$$

5.2.2 corr

With this tool we can calculate the correlation coefficient between every two features in a profile interval. We can pass the following arguments on the command line:

-a	Makes the tool analysing ARFF Files.
-d <directory path>	With this option we can specify the path to the data files.
-n <value>	This value determines the number of files which should be taken from the directory. Maybe the directory contains 100 files but we just want to take 10.

-O	Makes the tool analysing O-Cluster files.
-o <value>	Determines the offset *o* in the directory. The first taken NetFlow file will be the *o*-th one counting from the beginning.
-h	Prints a usage message.

The resulting files have the same filename like the analysed, but with an additional *corr* in the name. If no directory is specified, we can pass the input file as the last argument. In this case only this single file gets processed.

5.2.3 analyze

The **analyze** tool allows us to generate a statistic out of a couple of sud and/or corr files. We can either build a statistic for the correlation and the sud files only or generate a combined statistic. The combination calculates the FS-Algorithm from Section 4.2.3. We can pass the following arguments on the command line:

-c	This option sets the tool to analyse only .corr Files.
-d <directory path>	With this option we can specify the path to the data files.
-t <value>	Sets the correlation threshold. This threshold determines when a feature is considered as correlated with a other feature. The default value is 0.7
-h	Prints a usage message.

As an output we can get three different files. An *analyse_corr.txt* file when only corr files are analysed, an *analyse_sud.txt* for a single sud file analysis or an *analyse_complete.txt* for the FS-Algorithm. If no directory is specified, we can pass the input file as the last argument. In this case only this single file gets processed.

5.3 Clustering

5.3.1 bayes

This program adds the cluster identity for each host to the profile interval. The cluster identity for each host is written in a separate file generated by ODM's Bayesian Classifier. We can pass the following arguments on the command line:

-d <directory path>	With this option we can specify the path to the data files. The directory has to contain a equal number of profile interval files and cluster identity files. The identity file has to contain the substring *FileX* and has to end with *id.txt*, with *X* as the file number of the profile interval file it belongs to.
-h	Prints a usage message.

For every input pair we get a clustered profile interval. The file name is the same as the one of the original profile file including an additional *bayes_clustered* substring.

5.3.2 directclu

The **directclu** tool reads for every profile interval the belonging file with the cluster rules. This cluster rule files are generated by ODM. We can pass the following arguments on the command line:

-d <directory path>	With this option we can specify the path to the data files. The directory has to contain an equal number of profile interval files and cluster rule files. The cluster rule file name has to contain the substring *FilesX* and has to end with *rules.txt*, with *X* as the number of the profile interval file it belongs to.
-h	Prints a usage message.

For every input pair we get a clustered profile interval. The file name is the same as the one of the original profile file including an additional *direct_clustered* substring.

5.3.3 clustat

This tool calculates a cluster statistic for every clustered profile interval. It calculates several attributes like the cluster center, the average cluster radius and the standard deviation of the hosts to the cluster center. The host names of the three nearest hosts of each cluster are resolved. We can pass the following arguments on the command line:

-d <directory path>	With this option we can specify the path to the clustered data files.
-n	Normalizes the values before the distances are calculated.

-o <value>	This value specifies the outlier threshold (probability threshold) for the bayesian clustered files. If the probability of a host for belonging to a specific cluster is smaller than the threshold, it will be treated as outlier and not inserted into the cluster.
-h	Prints a usage message.

clustat produces for every input file a separate file with the cluster statistic. The file has the same file name including an additional *stats* substring. If no directory is specified, we can pass the input file as the last argument. In this case only this single file gets processed.

We can also check specific hosts towards their cluster affinity. If the working directory contains a file called *important_host_list.txt*, then for all hosts in the list the belonging cluster is searched. The host names are then attached next to the cluster statistic of the *stats* file. It is also possible to specify a whole subnet. The *important_host_list.txt* has to look like the following:

important_host_list.txt

```
129.132.46.11       www.ethz.ch           /* Single Host */
129.132.2.219       smtp.ee.ethz.ch
129.132.178.180     sip.ethz.ch
129.132.98.12       dns1.ethz.ch
129.132.208.0/16    public_vpn_subnet     /* Subnet */
.
.
.
```

5.3.4 cluextr

With the **cluextr** tool we can generate a separate file for each cluster containing all its active profiles. Such files are good for analysation or for generating plots. We can pass the following arguments on the command line:

-d <directory path>	With this option we can specify the path to the clustered data files.
-l	This option makes long filenames. The files have the same name with an additional substring *clstX*, with *X* as the cluster ID.
-t <value>	Objects, whose cluster belonging probability is smaller than this value are placed in a separate file with the substring *outlier* inside the file name. Only important for bayesion clustered files.
-h	Prints a usage message.

The tool produces for every input file multiple output files. The number of output file is dependent on the number of different clusters in the belonging profile interval. If the **-l** option is not set, the file names are *FileX_clstY.txt*, with *X* as the file number of the profile interval and *Y* as the cluster ID. If no directory is specified, we can pass the input file as the last argument. In this case only this single file gets processed.

5.3.5 clucomp

The **clucomp** tool compares two consecutive cluster statistic files generated by the **clustat** tool. If the firs file has the file name '*FileX...*' then there has to be another with the name '*File(X+1)...*'. We can pass the following arguments on the command line:

-d <directory path>	With this option we can specify the path to the cluster statistic files.
-h	Prints a usage message.

For each cluster in the first profile interval a corresponding cluster in the second interval is searched which could be most likely the same one. The tool saves the statistic in a separate file. The name of the output file has the form *FileX_File(X+1)_compared.txt*.

5.4 Profile Movement

5.4.1 profmove

This program calculates for every active host the distance the objects moves in the feature space between two consecutive time intervals. The tool can produce a database with the average moving distance of every host and the data base itself. The normal output file contains an overview of all active hosts between two consecutive time intervals and their moving behavior. We can pass the following arguments on the command line:

-d <directory path>	With this option we can specify the path to the profile data files.
-f	Ignores the features, which are written in the *ignore_features.txt* file for distance calculation.
-i	Reads a additional File called *important_host_list.txt*. For the hosts in this list an additional statistic will be printed in the output file.
-m	Saves a time statistic about the average moving of the database in a separate file called *mean_db_moving_history.txt*. If the *important_host_list.txt* is used, for these hosts a statistic is saved as well. These files are called exactly like the host name itself.

-o	Only creates and updates the data base and the time statistics, runs much faster. There are no overview files generated for consecutive time intervals.
-s	Creates or updates the data base. If there is no data base in the working directory the data base gets created. The file is called *movement_databse.db*. If the file already exists, then it gets updated.
-h	Prints a usage message.

This tool produces a couple of files. It saves a statistic into a file for every two consecutive time intervals called *FileX_File(X+1)_compared_profiles.txt* according to the input files. If the *important_host_list.txt* file is used, for every host in this file a separate statistic is printed into the *FileX_File(X+1)_compared_profiles.txt* file as well as a separate file with the moving behavior time statistic is stored. The *important_host_list.txt* file looks exactly the same as described in Section 5.3.3.

Profmove can also use a file called *ignore_features.txt*. This file determines which features are neglected during distance calculations. All features contained by the file are dropped. The form of the file looks as follows:

```
ignore_features.txt
num_sent_ip
num_diff_dest_ports
num_received_packets
num_recv_flows
.
.
.
```

5.5 Additional Tools

During the thesis, a lot of additional helpful tools have been developed to ease the usage of the files. The most important are explained in this Section.

5.5.1 normlize

This tool simply normalises all feature values between 0 and an upper value determined by the user. We can pass the following arguments on the command line:

-d <directory path>	With this option we can specify the path to the data files.
-u <value>	Defines the upper value of the normalisation. The elements are normalised between 0 and value. The default value is 1.
-h	Prints a usage message.

For every input file a corresponding output file gets created with the same file name including an additional *normalised* substring in the file name. If no directory is specified, we can pass the input file as the last argument. In this case only this single file gets processed.

5.5.2 filter

With the **filter** tool we can filter out designated profiles. For example we only want to consider profiles in a specific feature value range. The working directory has to contain a file called *filter_rules.txt*. We can pass the following arguments on the command line:

-c	Cuts off the header of the file, good for analysis in R [50]. The file name will be only *FileX.txt*.
-d <directory path>	With this option we can specify the path to the data files.
-n <value>	This value determines the number of files which should be taken from the directory. Maybe the directory contains 100 files but we just want to take 10.
-o <value>	Determines the offset *o* in the directory. The first taken NetFlow file will be the *o*-th one counting from the beginning.
-h	Prints a usage message.

The output file has the same name as the input file with an additional *filtered* substring. If no directory is specified, we can pass the input file as the last argument. In this case only this single file gets processed.

An example of the *filter_rules.txt* file can be see in the box following this paragraph. The first column determines whether the feature is active or not. A value of zero means that this feature is ignored during the filter process. The second column is the actual feature name the rule stands for. The following two columns describe the range in which a feature has to be in order to remain in the subset, otherwise the profile gets dropped.

filter_rules.txt

```
Active   Name                                   Min:      Max:

1        num_sent_ip                             0        0.3
1        num_rec_ip                              0.5      0.55
0        num_diff_src_ports                      0        1
.
.
.
```

5.5.3 rmfeat

This tool allows us to remove unnecessary features. This is useful when only a feature subset is considered. The program reads a file called *feature_table.txt* and removes the features according to the rules in this file. We can pass the following arguments on the command line:

-d <directory path>	With this option we can specify the path to the data files.
-n <value>	This value determines the number of files which should be taken from the directory. Maybe the directory contains 100 files but we just want to take 10.
-o <value>	Determines the offset *o* in the directory. The first taken NetFlow file will be the *o*-th one counting from the beginning.
-h	Prints a usage message.

The output file has the same name as the input file with an additional *topX* substring included. The *X* stands for the number of remaining features. If no directory is specified, we can pass the input file as the last argument. In this case only this single file gets processed.

The following box describes the *feature_table.txt* file, it has to contain all available features. The first column described whether the feature in the second column is active or not. If there is a value of 1, than the feature is active and gets not removed. Otherwise if there is a 0, it will get removed.

feature_table.txt

```
1       num_sent_ip
1       num_rec_ip
0       num_diff_src_ports
1       num_diff_dest_ports
0       num_diff_incoming_ports
.
.
.
```

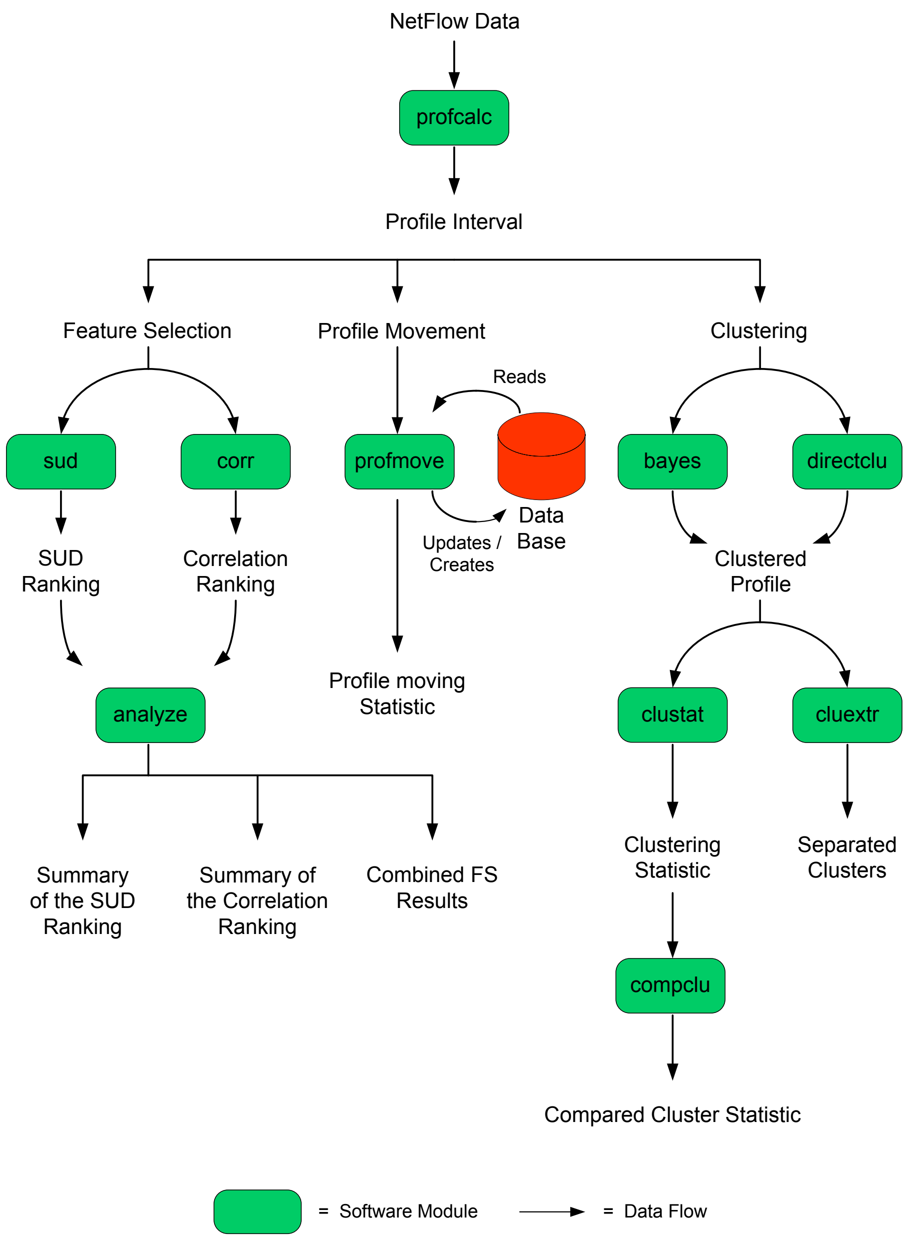


Figure 5.1: Software Tool Set

Chapter 6

Results

6.1 Building Host Profiles

In this thesis we only focus on the biggest ETH main subnet 129.132.0.0/16. This subnet can contain a maximum of 65536 different hosts and so we can only have a maximum of the same number of profiles. These profiles are built up over different time intervals. All flows captured in this interval are chosen for calculating the host profiles. The hosts which really appear in the actual interval are referred to as active hosts or active profiles.

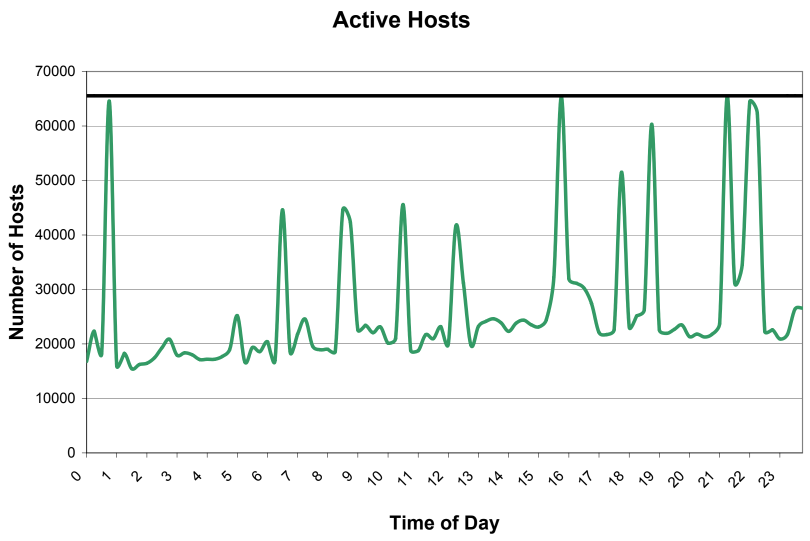


Figure 6.1: Active Host Statistic over one Day (Tue 2007-08-21)

The number of active hosts in an interval is highly unpredictable. It is not possible to compare the number of active hosts for the same time of day on different days. In Figure 6.1 we can see the history of the number of active hosts for one day. The interval length for counting the active hosts is fifteen minutes. The black horizontal line shows the maximal value of active hosts which is, as stated above, 65536.

We have never seen a profile interval with less than 10000 active profiles. This is the case, because there are a lot of hosts in the ETH net which are always online. This shows that there are a lot of hosts which offer different services to users, for example email or web servers.

The number of active hosts can change between consecutive intervals by a factor of four. This shows the high dynamic nature of the Internet.

6.1.1 Number of flows

In today's Internet we encounter huge amounts of data. In our case, we have to process up to one million flows every minute. When we compare the total amount of flows captured at the SWITCH border routers with the flows be-

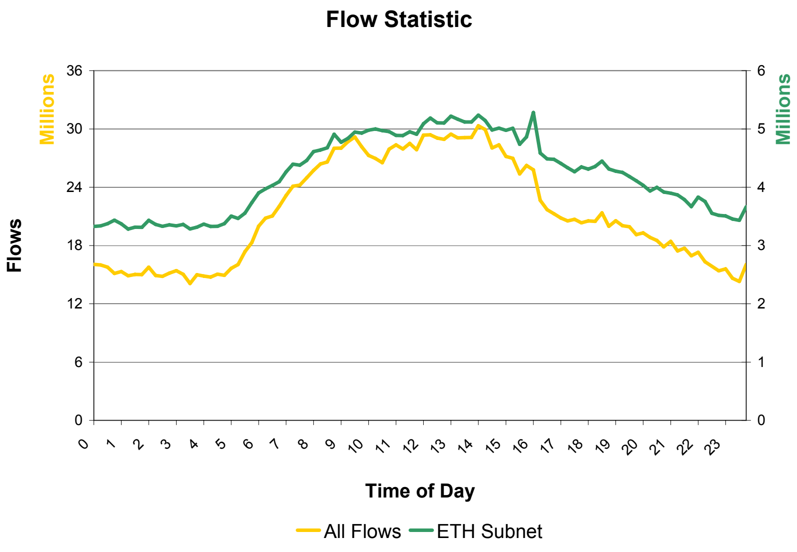


Figure 6.2: Average Flow Statistic over a Day

longing to our subnet, we can see that the ETH subnet behaves exactly like the whole SWITCH net (Figure 6.2). The left scale is for the whole SWITCH net, the right scale is for the ETH subnet. At night we see a fewer traffic than during the day. The traffic increases by 50% during daytime. We can clearly see the peek during the usual working time between eight o'clock in the morning and five o clock in the evening.

The data was monitored during the course of one week in August 2007. Out of this data the mean traffic volume for a day was calculated. The diagram shows the number of flows seen every fifteen minutes.

6.1.2 Calculation Speed

The calculation speed is mainly dependent upon three different parameters. Two of these parameters can be adjusted by the user. That is the interval length and the number of possible hash values. A deeper explanation about these hash values can be found in Section 4.1.2.1. It is obvious, that a short time interval is much more quickly calculated than a long one. The third parameter is the number of flows in the considered interval. This number is dependent on many factors, for example the actual time of the day the interval takes place and so on.

Table 6.1 compares the calculation time for different interval lengths with each other. For the hash Value h we chose once 997, which has empirically shown as good for short intervals, and for the second measurement we chose 1999, which is good for longer intervals. The general rule is, the longer the interval, the higher the hash value should be for fast execution.

	Calculation Speed			
	$h = 997$	$h = 1999$	num active profiles	speedup
5 min	33,4sec	31sec	10705	7,2%
15 min	1min 51sec	1min 28sec	22156	20%
60 min	10min 40sec	8min 18sec	60440	22,2%
2 h	22min 58sec	18min 32sec	65507	19,3%
12 h	2h 52min	2h 32min	65536	11,6%
24 h	9h 21min	undef.	65536	–

Table 6.1: Profile calculation speed[1]

As we can see from the Table 6.1, the hash value has a big influence on the calculation speed. For the 60 min interval, a hash of 997 is already too small. It appears, that always choosing a big hash value is a good solution. But this is not true for small intervals; a big hash value breaks the calculation speed down. In this case a lot of never used memory is allocated.

The memory allocation can become the biggest problem for calculating long intervals, because all the information is permanently held in memory. The calculation of the two hour interval with a hash value of 2999 requires more than the available memory of the target machine, which has a respectable memory of 8GB. There is also a trade off between memory usage and calculation speed. But as we have seen, a small hash value for a long interval still provides enough calculation speed compared with the length of the interval.

The twelve hour interval could be calculated in an acceptable time, but already the twenty-four hour interval causes some problems. With the small hash value, the amount of used memory is still acceptably small. But the problem is the *simple_list* construct; with such a small hash value, the search operation takes too long. Unfortunately, using the big hash value makes the calculation infeasible because of the memory swapping.

This memory problem comes from the *simple_list* structure, which is described in Section 4.1.2.1. With a hash value of 2999 we have 2999 different slots, which will be allocated for every list even, if they are probably not used. At least one slot will be used, otherwise the list will not be allocated. Every

[1]Calculated on a AMD Opteron 275 (2,2 GHz), 8GB RAM

host has up to five such lists and we have maximal 65536 active hosts. This means that initially already $5 \cdot 2999 \cdot 65536 \approx 1Mia$ pointers to such objects are allocated. Every slot-object pointer needs 8 bytes on a 64 bit machine and so this construction already needs nearly 8 GB of memory. Obviously, there are a lot of hosts which appear in very few flows, and so they have really sparse profiles and so many lists won't be allocated. But in a worst case scenario, this could happen.

Generally, the calculation speed is sufficiently fast for our purpose. It is no problem to use it in a real time application. Using the sliding window mechanism has no impact on the calculation speed. There are just some copy operations which can be neglected.

6.2 Feature Selection

As stated in Chapter 4, two different Feature Selection algorithms are used. As we will see later, especially the SUD algorithm has not delivered satisfactory results for our purpose.

6.2.1 Correlation

The difficulty with the correlation is the determination of a correlation threshold T_{xy} at which two features are considered as correlated. Figure 6.3 shows the

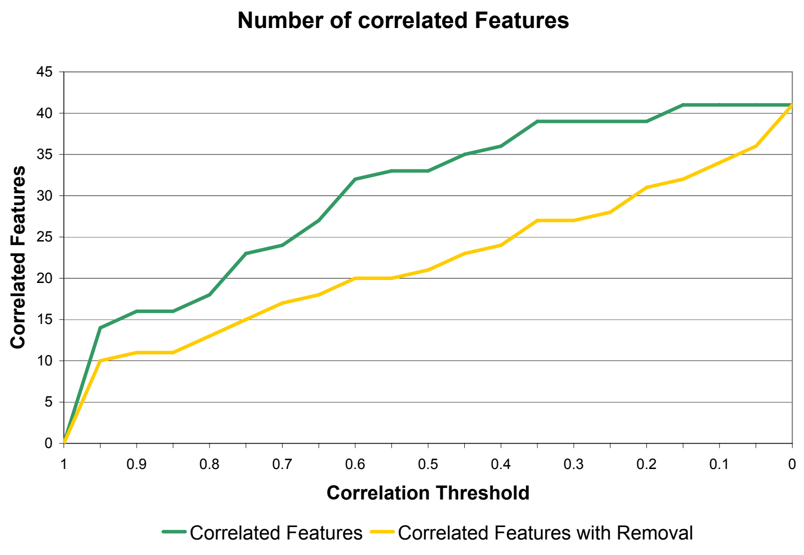


Figure 6.3: Number of correlated Features

number of correlated features depending on the used threshold. The data is calculated over one week with profile intervals of 15 minutes length. A feature is considered as correlated, when it has a correlation bigger than the threshold with one of the other features. This case is represented by the blue line (Correlated Features) in Figure 6.3. However, this line is a bit deceiving, because there are many features which are correlated with exactly one feature. Removing one of these features would make the other one uncorrelated. This case is shown with the red line (Correlated Features with Removal) in Figure 6.3. When a correlation between two features is found, we check if these two features correlate with other features too. The feature which correlates the most with all others is then removed and the other one stays 'active'.

Considering the blue line, fourteen features are really highly correlated ($r_{xy} > 0.95$). After this fast increase, the number of correlated features increases linearly with the decreasing threshold. Already at a threshold of $T_{xy} = 0.5$, three quarters of the features are correlated. With the red line, which is the important one for our purpose, we see that at a threshold of $T_{xy} = 0.5$ still half of the features are uncorrelated.

6.2.1.1 Highly correlated features

We now take a closer look at this fourteen highly correlated features. In Table 6.2 we can see this features and how they correlate:

Nr.	Feature	correlates with	Value
1	num_sent_ip	num_rec_ip	0.999
2	num_sent_flows	num_recv_flows	0.999
3	num_sent_tcp_flows	num_recv_tcp_flows	0.992
4	num_sent_udp_flows	num_recv_udp_flows	0.999
5	num_diff_dest_ports	num_not_well_known_dest_ports	0.999
6	num_diff_src_ports	num_diff_incoming_ports	0.998
7	num_diff_incoming_ports	num_not_well_known_inco_ports	0.999
8	num_diff_well_known_dest_ports	num_sent_udp_flows	0.993

Table 6.2: Correlation table

Here are some explanations as to why these features are that highly correlated:

Case 1: A communication between two hosts is mostly bidirectional and so two flows are generated. One from the sender to the receiver and one in the other direction. For this reason, the number of different contacted IPs is generally exactly the same as the number of different received IPs.

Case 2: Here the explanation is exactly the same as in Case 1, most of the time we have bidirectional communications.

Case 3: Initiating a TCP [2] connection by a sender to a receiver automatically causes a connection from the receiver to the sender, too. It generates a virtual channel between this two hosts. And so these two variables have to be highly correlated.

[2]**T**ransmission **C**ontrol **P**rotocol

Case 4: UDP [3] is a connectionless protocol, no channel is set up. For this reason it is not obvious why these two features are highly correlated. But today's application built upon UDP are mostly bidirectional, like VoIP [4] and likewise.

Case 5: In this case we see that most of the initiated connections by a host are not well-known services [5]. This correlation also says, that a user uses only a small number of well-known services at a time but has a lot of connections to other hosts on not well-known ports as well.

Case 6: This case is more difficult to understand. It is again a fact of the mostly bidirectional manner of communications. The number of different source ports describes, broadly spoken, the number of different used services. All these services respond to a different port, so these two numbers are mostly equal.

Case 7: This case has a lot in common with Case 5. Most of the hosts are computers of 'normal' users, like laptops etc. This host does not often offer some well-known services like FTP or Web servers. For this reason most of the incoming port numbers lie outside the well-known range.

Case 8: This is the most special case. It appears that most not well known services uses the UDP protocol for transmission.

This correlation values shows that a lot of features have more ore less the same meaning. Mostly this high correlation is not directly obvious at first sight.

6.2.1.2 Interval length

The interval length does not affect the correlation coefficients at all. An interval length of one minutes already delivers the same results as an interval of 60 minutes. The results can be seen in Figure 6.4. Similar to the red line in Figure 6.3, one of the correlated features is removed and the other one stays in the set of active features.

6.2.2 SUD

As written in Section 4.2.1, SUD uses an entropy measurement to determine the important features. The entropy is generally high if we do not have uniformly distributed data points. With our currently used 41 features the maximum euclidean distance between two different data points is:

$$\sqrt{x^2 + y^2 + z^2 + \dots} = \sqrt{1 + 1 + 1 + \dots} = \sqrt{41} = 6.4$$

Because of the normalisation, the maximum distance in one dimension can be 1. But the average distance in our data between the data points lies at about 0.9. This shows that a lot of points are near each other and lie in the same region of the 41-dimensional hyper cube. There is no other region with a lot of members far away. We can see the same thing when we investigate the histograms of the features (See Figure 6.6). They are really concentrated on one typical region.

[3]**U**ser **D**atagram **P**rotocol

[4]**V**oice over **I**nternet **P**rotocol

[5]Well-known means, that we connect to a port number less or equal to 1024, on these ports typical services are defined. For example port 80 is for HTTP access (Websites)

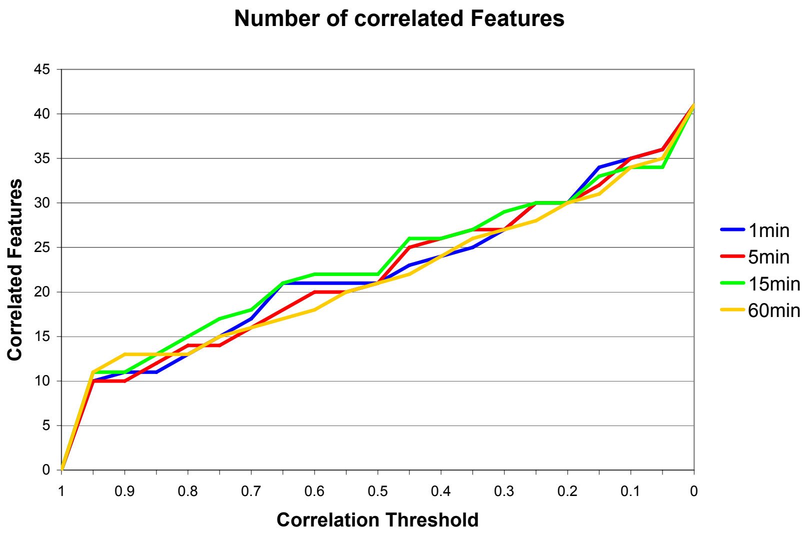


Figure 6.4: Number of correlated Features depending on the Interval Length

6.2.2.1 Calculation using Subsets

The SUD algorithm is extremely costly to calculate as we have seen in Section 4.2.1.3. It has a quadratic time complexity with the number of profiles and the number of features as well. For this reason, the entropy was calculated only on a subset of the profiles. Table 6.3 shows a statistic about the accuracy of the results depending on the size of the subset.

	Number of Active Profiles		SUD Accuracy	
	≤ 35000	> 35000		
Subset size	$\overline{D}$	$\overline{D}$	All	least 15
1%	1.164	0.722	43.9%	75%
5%	0.96	0.636	48.8%	86.7%
10%	0.905	0.583	51.2%	86.7%
20%	0.879	0.565	53.7%	86.7%
50%	0.857	0.548	58.5%	93.3%
100%	0.857	0.543	100%	100%

Table 6.3: SUD Subset Statistic

The statistic about the average distance was calculated with 800 profile intervals of each type. That is to say 800 intervals with less than 35000 active profiles and with 800 intervals with more than 35000 active profiles. The actual time of day of these profile intervals is equally distributed over one week. The interval length amounts to fifteen minutes. For reasons of accuracy, 20 profile intervals have been taken, again with an interval length of fifteen minutes.

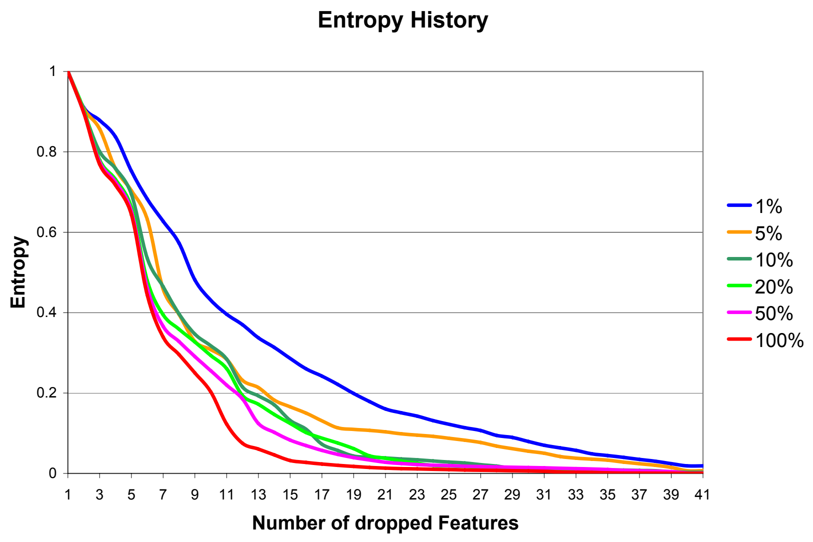


Figure 6.5: Remaining Entropy depending on the Number of dropped Features

The accuracy of the SUD ranking decreases strongly when using a subset. A subset size of 50% already decreases the accuracy by about 40%. But if we only consider the fifteen least relevant features, we see that the accuracy reduction is much less. Already a small subset can classify the least relevant attributes correctly. The reason why the accuracy subsides for the other features is because of their distribution. All of them have a very similar, heavy-tailed distribution. To illustrate such a heavy tailed distribution, Figure 6.6 shows the histogram of the *max_sent_flow_length* feature compared with a less heavy-tailed histogram of *average_recv_packets_length*. Such a heavy-tailed distribution guarantees a small entropy value in the SUD algorithm. A small change in the distribution can already cause a big change in the SUD ranking. Because their entropy values lie really close to each other. This circumstance can be seen in Figure 6.5. The red line represents the whole subset and so the real entropy progression.

After fifteen dropped features the entropy for the whole data set is already nearly zero, so the information content of every of the remaining 26 features is nearly equal. By choosing a smaller subset, the entropy decreases more slowly. But even with a small subset of one percent of the active hosts, the real behavior is well approximated. After dropping the fifteen least important features, the uncertainty has dropped to 0.28. Compared with the value of 0.03 for the whole data set, it is still quite high. Nevertheless, as we can see from Table 6.3 the accuracy of 75% of the SUD ranking is still good compared with the size of the subset. A subset size of ten percent of the original data set can be considered as big enough for getting reliable results. The entropy curve is already really close to the one from the whole data set and most of the features are ranked correctly. In the following experiments, a subset of ten percent is always chosen.

A ten percent subset can cause, at maximum, 6500 active hosts. Because of the high complexity of the SUD algorithm, the calculation can still take a long time. On the test machine[6], the calculation took 2 hours and 7 minutes when using all four possible cores. On the other hand, the feature selection process has to be done only once.

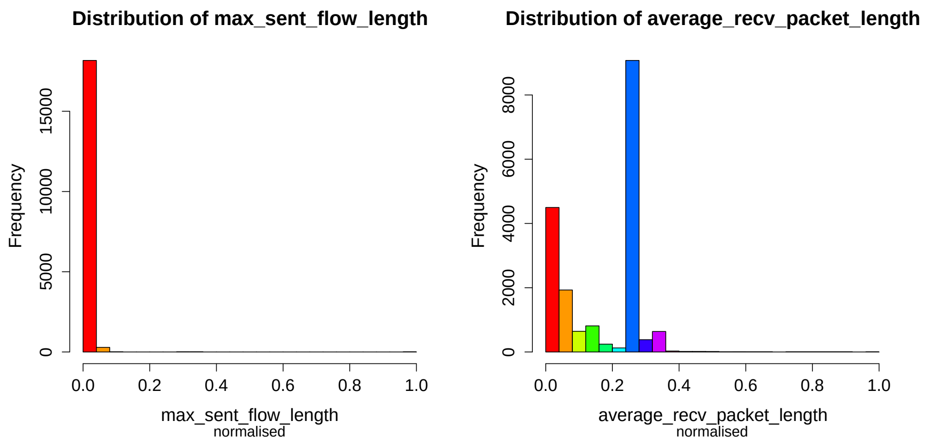


Figure 6.6: Heavy-tailed vs. less Heavy-tailed (15 Minutes Interval)

Another interesting point is the average euclidean distance $\overline{D}$ between the data points. For this we have to divide the intervals into those with many active profiles (> 30000) and those with less active profiles (≤ 30000). We can see from the table, that this average distance decreases with an increasing subset size and also with an increasing number of active profiles.

The reason why the average distance decreases with an increasing amount of active profiles is because of the 'nature' of the active hosts. There are a lot of 'special' hosts which appear in every time interval, for example the ETH web server or the DNS[7] servers. On the other hand, the majority of the hosts appear very infrequently. A large part of this majority consists of general-purpose hosts like student laptops and the like. These kind of hosts mostly have a very similar communication behavior and lie really close to each other in the feature space. On the other hand, the 'special' host differs quite strongly in the communication behavior and therefore they lies far away in the feature space from these general-purpose hosts. So if the number of general-purpose hosts increases, the average distance decreases because they dominate the whole population with their short inter host distance.

6.2.2.2 Different Interval Length

Here we check to see whether the interval length has an impact on the SUD ranking. Probably the SUD algorithm rates the importance of the features totally differently. A longer or shorter interval could result in a totally different feature ranking. For this reason, we take our fifteen minute interval as refer-

[6]2x Opteron 275 (2.2 GHz, Dual-Core), 8GB RAM

[7]Domain Name System

ence and compare the fifteen least important features with those of the other intervals. The results can be seen in Table 6.4.

Interval Length	Accuracy of least 15	Entropy after removing
1min	90%	0.44
5min	93%	0.25
15min	100%	0.17
60min	88%	0.11

Table 6.4: SUD with different Interval Length

The results show that the ranking of the feature is not affected by the interval length. All of the compared time intervals classify most of the top fifteen least important features equally. A one minute interval is already enough for an accurate ranking. But the uncertainty with a value of 0.44 is still much bigger after removing the fifteen least important features as in the other time intervals. This shows that the distribution of the features is much less heavy-tailed than in bigger intervals. Generally speaking, the shorter the interval, the slower is the decline of the entropy (Figure 6.7). To compare the difference in the distribution with the fifteen minute interval, in Figure 6.8 the histograms of the same features are plotted as in Figure 6.6. We can see the same tendency, the left distribution is still heavy-tailed, but more moderate than in the fifteen minutes interval. The same observation can be made with the right distribution, the data points are more scattered than in the longer interval.

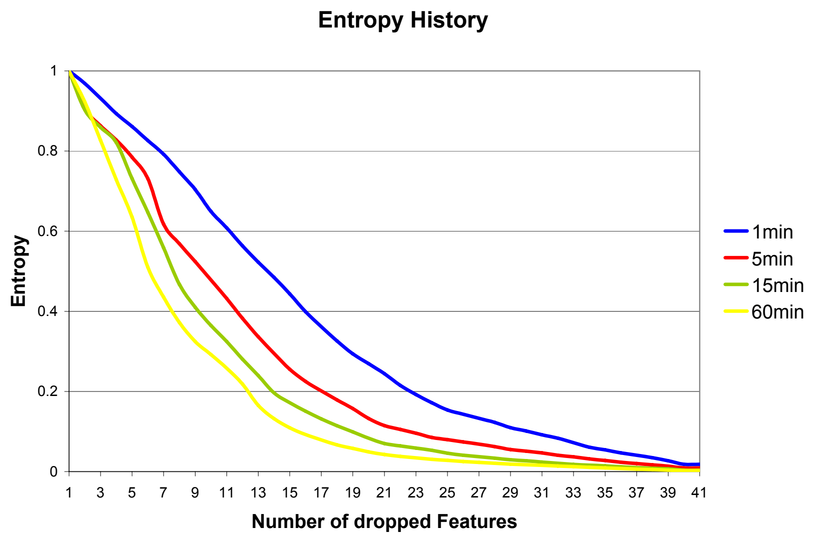


Figure 6.7: Remaining Entropy depending on the Interval Length

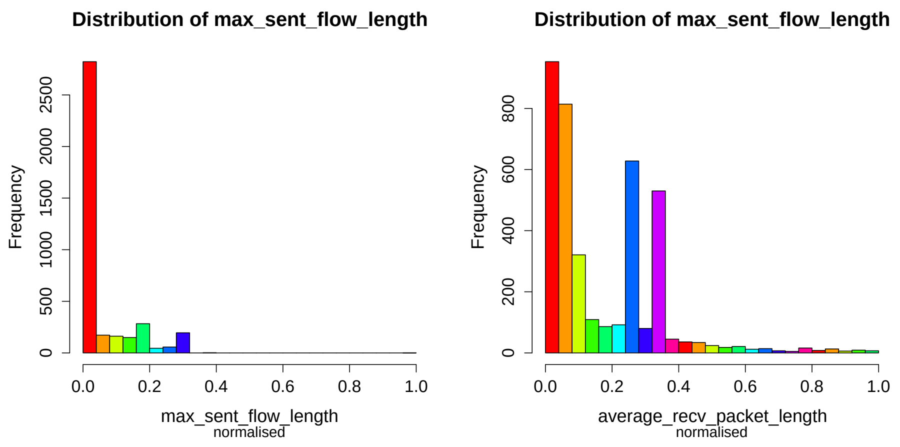


Figure 6.8: Heavy-tailed vs. less Heavy-tailed (1 Minute Interval)

6.2.3 FS Algorithm

For the actual feature selection, we combine the correlation with the SUD algorithm. A more precise description of the used algorithm can be found in Section 4.2.3.

The SUD algorithm does not recommend a specific number of features to use. But from the entropy figures we see that the last fifteen features are much less important than the other 26 features. But a lot of these important features can be correlated, and so we have to eliminate them as well.

First of all, we have to determine a correlation threshold T_{xy}. It describes at which value two features are considered as correlated. Generally, we can divide the correlation value into different regions:

Correlation	Interpretation
$r_{xy} = 0$	not correlated
$0 < r_{xy} \leq 0.5$	weak correlated
$0.5 < r_{xy} \leq 0.8$	medium correlated
$0.8 < r_{xy} \leq 1$	strong correlated
$r_{xy} = 1$	perfect correlated

Table 6.5: Interpretation of the Correlation Coefficient

We do not want strongly correlated features, so for this reason the correlation threshold should lie beneath 0.8. A low medium correlated or even a weak correlated value is preferable.

The SUD algorithm classifies heavy-tailed distributed features as more important than the other ones. But generally, heavy-tailed distributed features are much more correlated among themselves. This leads to the problem, that the resulting features after the FS algorithm mostly belong to these fifteen least important SUD features. This is because the FS algorithm separates in a first step according to the correlation coefficient and in a second step according to the SUD ranking. The so found feature subset is now called the **FS Set**.

To avoid this problem, we eliminate those features in the final ranking, which belong to these fifteen least important features. To determine these fifteen features, we take the average SUD ranking of the fifteen minute interval and choose the fifteen least important ones.

The number of features in the final ranking can be found in Table 6.6, one column with the original FS final ranking (FS Set) and another with the elimination of these least fifteen from the final ranking (FS Set reduced). In all

Correlation	Number of Features	
Threshold T_{xy}	FS Set	FS Set reduced
0.4	16	7
0.5	18	7
0.6	19	8
0.7	23	10

Table 6.6: Final Ranking dependent on the Correlation Threshold T_{xy}

cases, more than half the features in the final ranking belong to these fifteen least important features. Choosing a correlation threshold of 0.6 reduces the final feature set to eight remaining features. This set is now called **FS Set reduced**. These most promising features are listed in Table 6.7:

Rank	Feature Name
1	*num_sent_bytes*
2	*num_sent_flows*
3	*num_recv_bytes*
4	*num_sent_tcp_flows*
5	*average_sent_packets_per_flow*
6	*min_recv_Packets_per_flow*
7	*min_recv_flow_length*
8	*min_sent_packets_per_flow*

Table 6.7: Final Feature Subset ($T_{xy} = 0.6$)

6.3 Clustering

The clustering process is done by using different feature sets. We have taken fifteen minute time intervals with a sliding window size of three. This means, for consecutive profile intervals, the interval is slid by five minutes.

ORACLE makes some suggestions on how to prepare the data before clustering in order to get good results:

- Eliminate outliers by Trimming instead of Winsorizing.
- Discretisation of the numerical values, because O-Cluster is not a distance based algorithm, otherwise an optimal splitting is not possible.
- Normalisation is not necessary, because the cluster are not built on top of distance measurements.

The clustering result is highly dependent on the number of active profiles inside an interval and also on the actual time of day. The number of resulting clusters is determined by the sensitivity factor ρ. The parameter setting of the O-Cluster Algorithm will be the same for all experiments unless some other value is stated. The default value for the Sensitivity parameter will be $\rho = 1$.

We will always investigate intervals with three different number of active profiles: Some intervals with 20000 active hosts, some with 40000 and some with 60000. The time of day of the chosen intervals are taken randomly.

6.3.1 The FS-Feature Set

The clustering is done by using the most promising features chosen by our feature selection algorithm. As we have seen in Section 6.2.3, for a correlation threshold $T_{xy} = 0.6$ we have 19 remaining features in the final ranking or, after eliminating the fifteen least important, only 8 residual features.

6.3.1.1 FS Set reduced (8 Features)

When we cluster different intervals by using the eight feature of Table 6.7, we can see that O-Cluster only uses three of the features at most. We have to remember that the O-Cluster algorithm always splits the space along a specific axis. Only this three features separate the space well enough in order to form clusters. For this reason we can only keep these three features in order to get the same clustering result.

Only the intervals with 40000 active profiles need all of the three features, as we can see in Table 6.8. From the number of splittings, the *min_recv_flow_length* feature seems to be the most separating one. The fluctuation of the number of clusters increases with the number of active profiles. A lot of active profiles do not guarantee a lot of different clusters.

Looking only at these three dimensions shows us the well known fact, that our features have a heavy-tailed distribution. The distribution of the data points in this three dimensional space can be seen in Figure 6.9. Most of the data points are concentrated in a small part of the space. There is no obvious cluster visible. Even when we zoom in (right diagram of Figure 6.9) we cannot detect any cluster. The clusters found by O-Cluster are not directly obvious, we can see them in Figure 6.10.

The probability threshold is needed with the Bayesian Classifier of the ODM, because every data object has a determined probability for every cluster, which describes the probability for the data object that belongs to this cluster. The probability threshold describes at which value an object is considered as an

	Number of active Profiles		
	20000	40000	60000
Feature	Number of Splittings (Max)		
num_recv_bytes	1	2	2
average_sent_packets_per_flow	-	1	-
min_recv_flow_length	5	6	8
	Number of Clusters		
Min	5	6	2
Max	7	10	11

Table 6.8: Clustering Overview

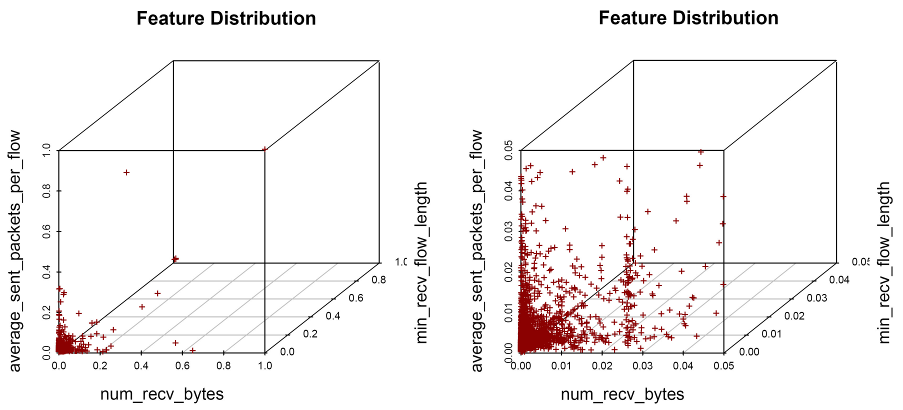


Figure 6.9: Distribution of the Data Points (FS Set reduced)

outlier. This is because the probability for the most promising cluster is smaller than the threshold. In the left diagram of Figure 6.10 the threshold is set to one, which means that only objects which can be assigned to a cluster with 100% certainty are put into the cluster. The data points colored in light blue are those which could not be assigned to a specific cluster, the so-called outliers. In the right diagram, the threshold is set to zero, which means that every data point is put into that cluster which it belongs to most likely.

Every color describes a different cluster. We can see from the picture that nearly no splitting has taken place along the *average_sent_packets_per_flow* axis. All the clusters career along this axis. In order to illustrate the clustering better, we take a look at the *average_sent_packets_per_flow* axis of the clustering. We can see now a two dimensional slice from the feature space. The slice is depicted in Figure 6.11. The black lines describe the clustering derived from the cluster rules of O-Cluster. In order to see the cluster regions, the image is zoomed to the interesting part of the space, as is shown in the images of Figure 6.10. We

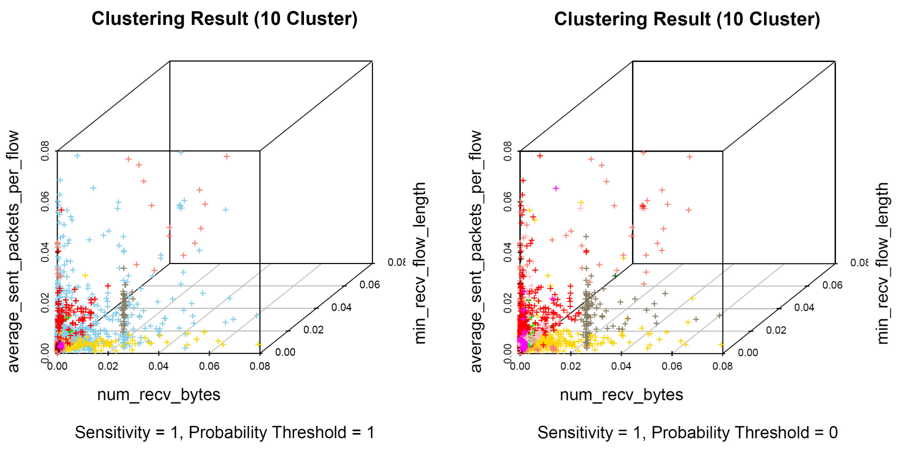


Figure 6.10: Cluster Solution (Classified with the Bayesian Classifier)

can clearly see the axis parallel splitting rules of O-Cluster.

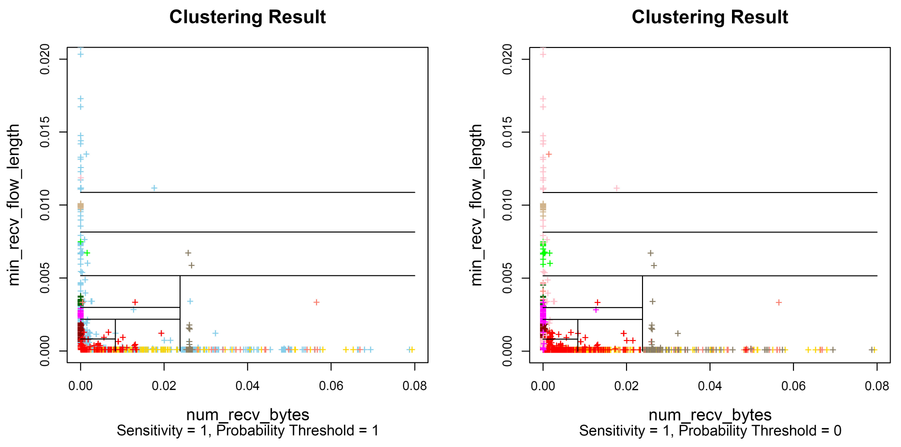


Figure 6.11: Cluster Regions

In the left diagram, only data points are colored with the cluster color if they belong to one cluster for sure (cluster probability = 1). But we can still see some data points which are colored with the same color in different cluster regions. This has something to do with the Bayesian Classifier.

If we take the cluster rules delivered by the ODM, and generate the clustering out of it we get well separated clusters. This is possible, because our data set is small enough to fit completely into the input buffer of O-Cluster. The rules are enough to cluster every data object. We have to remember that the Bayesian Classifier is for the case that not all data points fit into the cluster. In such a case, some information probably got missed. The clustering result generated out of the cluster rules can be found in Figure 6.12.

We can still see some data points which look as if they spread over different cluster regions. But this is just the case when we are looking into two dimensions

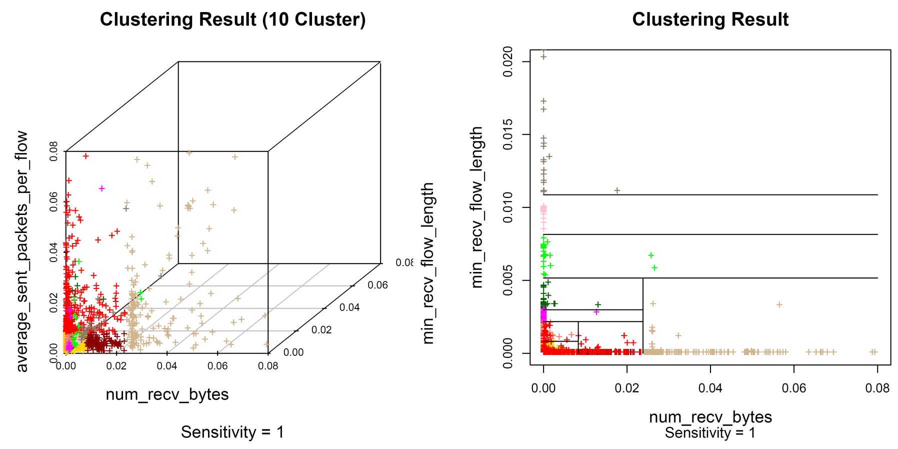


Figure 6.12: Cluster Solution (Classified with Direct Rule Classifier)

of a three dimensional clustering. The clusters, which spread over the clustering borders are clustered along the third, not visible axis.

To illustrate the difference between the Bayesian Classifier and the Direct Rule Classifier, we generated a real two dimensional clustering out of the same dimensions. The results are presented in Figure 6.13. We can see that the

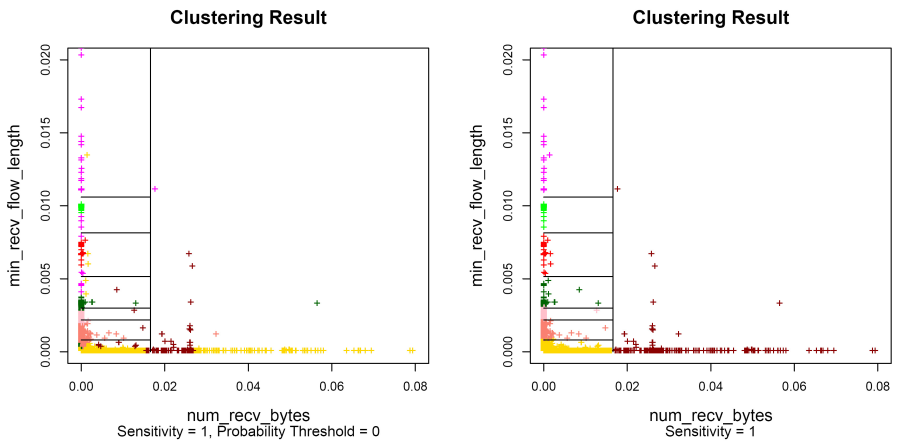


Figure 6.13: Bayesian Classifier vs Direct Rule Classifier

O-Cluster algorithm separates the axis at other points than in the three dimensional case.

The number of members of each cluster differs strongly. There are a lot of small clusters with a few members. A statistic can be found in Table 6.9. In only two clusters more than 90% of the hosts are concentrated. All other clusters are very small, with sometimes only a couple of dozen hosts.

The interpretation of the found result is very difficult, because the clusters are not well separated. It makes no difference if we have clustered by using the Bayesian Classifier or by the Direct Rule Classifier. Maybe using only eight

	Bayesian Classifier		Direct Rule Classifier
	Probability Threshold		
	1	0	
Cluster ID	Cluster Members		
1	1 (0,002%)	97 (0.23%)	51 (0.12%)
2	34 (0.08%)	90 (0.21%)	196 (0.46%)
3	17428 (41.17%)	17879 (42.34%)	17902 (42.29%)
4	76 (0.18%)	148 (0.35%)	119 (0.28%)
5	93 (0.22%)	375 (0.89%)	340 (0.80%)
6	116 (0.27%)	478 (1.13%)	118 (0.28%)
7	20967 (49.53%)	21740 (51.35%)	22201 (52.44%)
8	1208 (2.85%)	1385 (3.27%)	1254 (2.96%)
9	2 (0.005%)	74 (0.17%)	82 (0.19%)
10	64 (0.15%)	68 (0.16%)	71 (0.17%)
Outliers	2345	0	0

Table 6.9: Clustering Statistic

features, of which only three are chosen by O-Cluster to produce clusters, are not enough for a good result. It is not possible to generate a taxonomy out of this cluster, we could not assign a typical behavior to each cluster, for example web servers or the like. In these two big clusters, any kind of host can be found.

As the authors of the O-Cluster algorithm have stated in their paper, O-Cluster performs a better clustering the higher the feature dimension is.

6.3.1.2 FS Set (19 Features)

We now take the feature set of the final ranking for the clustering process. For our correlation threshold $T_{xy} = 0.6$, this are 19 features. Again, not all feature are used by O-Cluster to generate the clustering. At the maximum, 13 features are taken by the algorithm to form the clusters. There were always the same 13 features, and if less features had been chosen, it would have been a subset out of these 13 features. This leads to the conclusion that six of these features from the final ranking are that heavy-tail distributed, and the algorithm cannot separate along their axis. If only these six features are chosen for clustering, the result would be one big cluster.

The number of resulting clusters is highly dynamic, but generally, the more active profiles an interval has, the more clusters result. A clustering summary can be found in Table 6.10.

We can see that the average number of used feature with an increasing number of active profiles decreases. This shows the fact that with more profiles the feature distributions become more heavy-tailed. We could observe the same

		Number of active Profiles		
		20000	40000	60000
Number of used Features	Max	13	12	11
	Min	11	9	10
	Average	11.6	10.5	10.3
Number of found Clusters	Max	74	133	124
	Min	51	47	95
	Average	61.7	101.4	110

Table 6.10: Clustering Overview

behavior with the interval length: The longer the interval, the more heavy-tailed the feature distributions. Heavy-tailed distributions makes it much harder for the O-Cluster algorithm to distinguish between dense regions and thus to find clusters.

We observe here the same fact as in the analysis with 8 features. There are a few really big clusters with about 90% of the active profiles and a lot of really small clusters. Again, we could not assign a behavior to every cluster which can describe the kind of host contained by the cluster. Therefore, it makes no difference whether we classify the objects by the Bayesian classifier or by the Direct Rule Classifier. No specific behavior could be extracted for each cluster.

6.3.2 The whole Feature Set

In this section we cluster different intervals with the whole feature set, because we have seen that the features derived from feature selection have not lead to satisfactory results.

The number of resulting clusters does not increase strongly anymore. There has never been a case in which O-Cluster used more than sixteen features for building the cluster rules. This indicates that the newly inserted features do not put more information into the system. The algorithm cannot split along most of these axis. Therefore, we cannot improve our clustering anymore. This shows that the feature selection process could extract the most important features.

Out of our 41 different features, only 22 have been used at least once by O-Cluster to build the model. These features can be found in Table 6.11 with their mean usage probability for clustering.

These values are calculated with a interval length of fifteen minutes for the profiles. The number of active hosts span between 20000 and 65000 and the time of day is picked by random.

We made again the observation, that the number of used features decreases when the number of active profiles increases. The explanation is the feature distribution, which gets more heavy-tailed with an increasing amount of data objects. The number of used features span between sixteen at a maximum and twelve at a minimum.

Used Features for Clustering and its Usage Percentage			
most_received_host	100%	*max_recv_packets_length*	100%
min_recv_flow_length	100%	*most_incomminc_port*	100%
most_connected_host	100%	*most_connected_port*	100%
average_recv_flow_length	90%	*min_sent_packets_length*	90%
min_recv_packets_length	75%	*min_sent_flow_length*	75%
max_sent_packets_length	75%	*average_sent_flow_length*	75%
average_recv_packets_length	65%	*num_recv_bytes*	45%
max_sent_flow_length	45%	*average_sent_packets_length*	45%
max_recv_flow_length	35%	*average_recv_packets_length*	25%
min_sent_flow_length	25%	*num_sent_packets*	15%
num_received_packets	15%	*min_recv_packets_length*	15%

Table 6.11: Features used by the O-Cluster Algorithm

6.3.3 Parameter Settings

6.3.3.1 Sensitivity Parameter ρ

The O-Cluster Algorithm generally needs one Parameter, which is the Sensitivity ρ. As we have seen in Section 4.3.3, this parameter influences the number of resulting clusters. Now we compare how the actual cluster number is affected by different settings of ρ. For the experiments, we use all 41 features for clustering.

We observed, that the number of clusters can vary highly between different intervals. Even if we compare intervals with the same length, same number of active profiles and with the same value for the sensitivity ρ. But the behavior of the characteristic of the curve is for all intervals the same. For this reason we show an example of a profile with 20000 active Hosts. The behavior curve in Figure 6.14 can stand for all possible cases. Because of that, we use a percentage measurement for the y axis. It describes the number of possible clusters found. 100% describes the case, when the sensitivity is set to $\rho = 1$ and all clusters are found.

At the beginning, the number of found clusters increases linearly with an increasing sensitivity. The moment the value has passed 0.5, the number of clusters increases exponentially. With a sensitivity of $\rho = 0.5$ only 20% of the possible clusters are found.

6.3.3.2 Number of Features

Here we want to show, how the cluster number behaves with different number of features. For the experiments, we choose a sensitivity of $\rho = 1$. The number of clusters is very dynamic, it is nearly impossible to compare between different clustering. For this reason we made an experiment with one interval and removed every step the feature, along which O-Cluster had split the fewest of all.

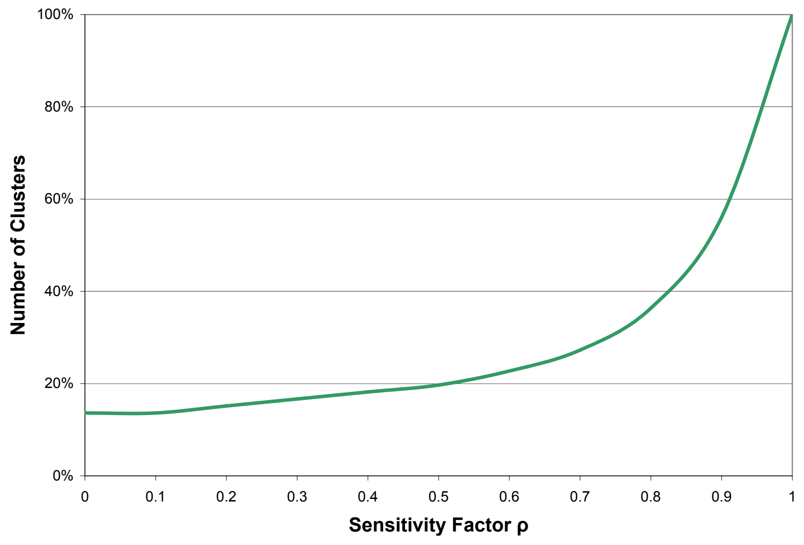


Figure 6.14: Number of Clusters dependent on the Sensitivity ρ

In Figure 6.15 wee see an example for the development of the number of cluster dependent on the number of features for clustering. Both axis are measured in percentage. Because the number of used features and found clusters differ between intervals.

We can see that the number of found cluster can again increase when we remove some features. This show the fact, that too many features can reduce the clustering quality. Furthermore we can see that a few features are already enough to find most of the clusters. Already a clustering with 50% of the features can find more than 80% of the clusters.

6.3.4 Time Calculations

We compare the time ODM needs to cluster intervals with different number of active profiles and different number of features. The results can be found in Figure 6.3.4. We have to remember that the time complexity of the clustering algorithm is $O(nd)$, with n as the number of data objects and d as the number of features respectively dimensions.

The diagram shows that the linear time complexity proves well-founded. The slope of the curve describing the calculation time by using different number of features rises slower than the other curve. Therefore it is more costly to double the number of features than the number of profiles.

6.4 Profile Movement

We have investigated the movement of each profile between two consecutive time intervals. We used again the three different feature sets. That is the one containing all features, the feature set delivered by the feature selection process

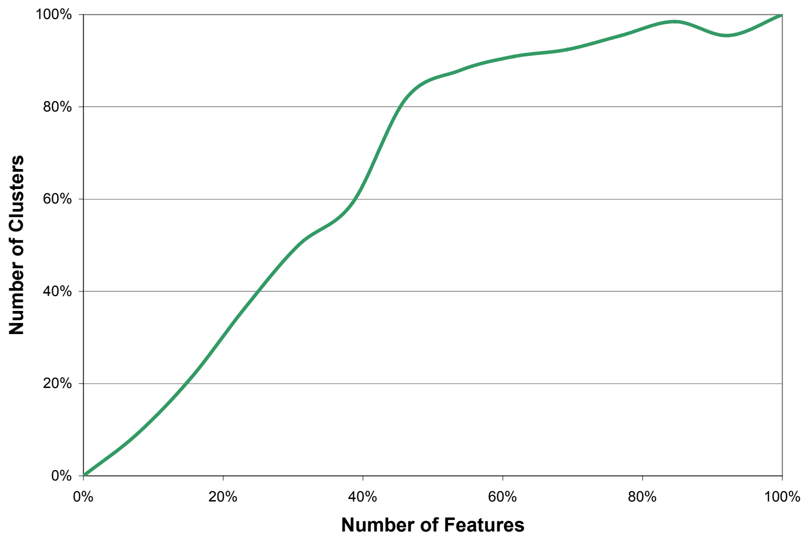


Figure 6.15: Number of Clusters dependent on the Number of Features

(FS Set) and the subset of the set delivered by the feature selection (FS Set reduced).

We calculated the moving distance of each host among fifteen minutes intervals over the course of one week. In Table 6.12 a statistic is printed, the average moving E is here the average moving distance of all active hosts in a profile interval. All values had been normalised before calculation. In order to compare the results we introduce the Moving Factor. Because the different number of dimensions makes it impossible to compare among the moved distance. In this factor the dimensionality is removed. The Moving Factor κ is actually the same as the average moving percentage and calculated as follows:

$$\kappa = \frac{E}{\sqrt{d^2}} \tag{6.1}$$

Feature Set	Average Moving E	Moving Factor κ	Moving Percentage
Whole Set	0.343	0.0536	5.36%
FS Set	0.335	0.077	7.68%
FS Set reduced	0.028	0.0098	1%

Table 6.12: Profile Moving Statistic

We can again see the influence of the feature distribution. If there are many heavy-tailed features involved, than the Moving Factor κ is as low as in the reduced FS Set. But generally, the profiles do not move a long distance. The most dynamic set is the FS Set, the profiles move 7.68% on average between

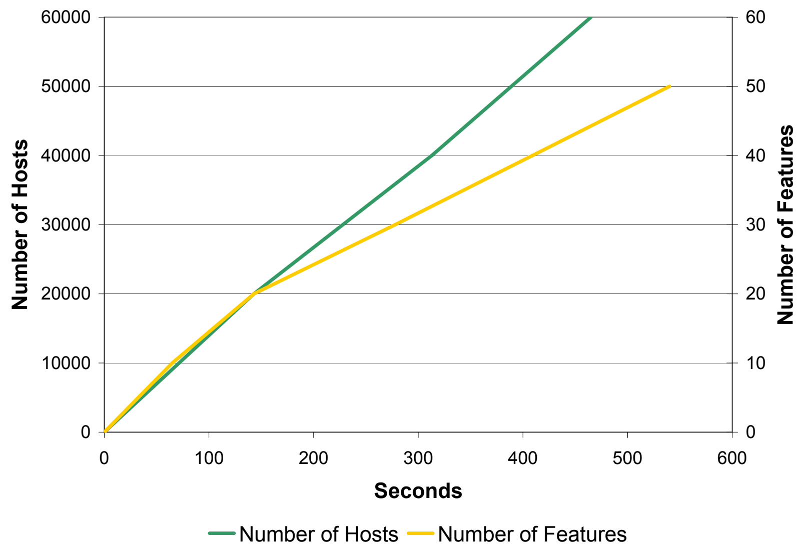


Figure 6.16: Performance of O-Cluster

Calculated on a Pentium M 1.8 GHz with 1GB RAM.

two intervals. In the reduced FS Set they move only 1% on average.

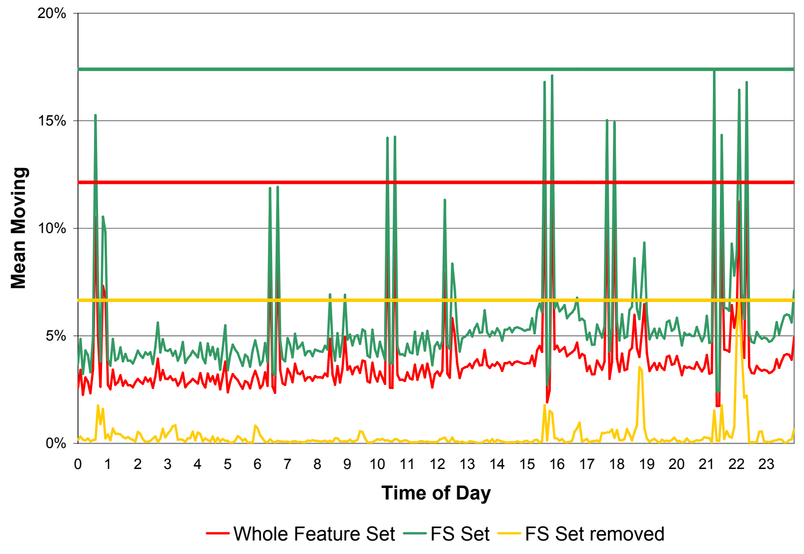


Figure 6.17: Average Moving of the Profile Interval during a Day

When wee take a look how the mean moving evolves during a day, we see that the values can makes big jumps (Figure 6.17). By taking the whole feature set, the data objects can change their positions by more than 10% at maximum. When we only take the FS Set, than the moving distance increases up to more than 15%. This is a result of the fact, that we find a lot of these fifteen least important SUD features. Their distribution is not as heavy-tailed as those of the other features. For this reason, the profiles move further distances. The thick horizontal lines in the figures denotes the peak value of each subset.

To flatten this peaks, we can take the FS Set reduced. This behaves much more static than the other feature sets. We can clearly see in Figure 6.18 how the standard deviation decreases compared to the other two sub sets.

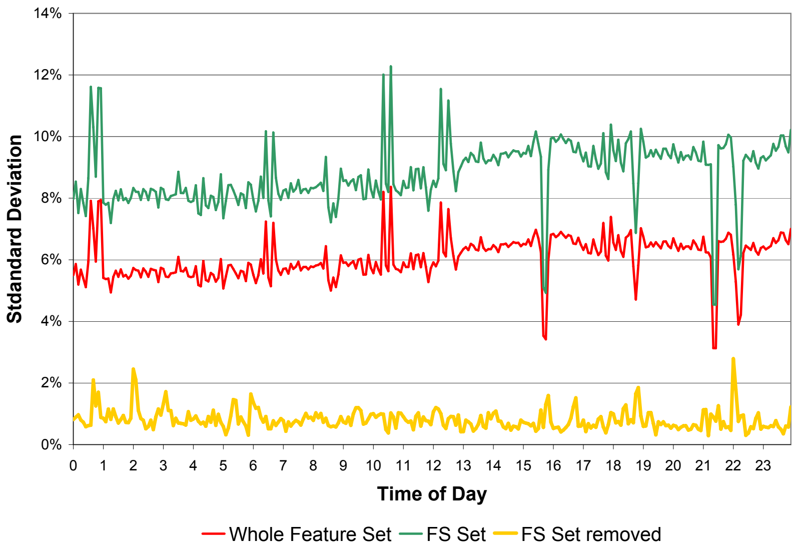


Figure 6.18: Standard Deviation of the Profile Interval during a Day

6.5 Anomalies

Now we want to analyse some anomalies and investigate their impact on our system. Can our methods detect such abnormal behavior. Firstly we take a look at some single abnormal hosts and secondly we investigate the impact of the Blaster outbreak [51].

6.5.1 Malicious Host

In this section we check the behavior of single malicious hosts. The SWITCH security team delivered us with information about hosts which have showed a suspicious behavior. For this purpose we investigate the distance the host jumps between two consecutive time intervals.

In Section 6.4 we saw that the moving behavior is highly unpredictable. The best results were achieved by using the FS Set reduced. For this reason we will use this feature subset for our investigations.

We consider the week between Monday 20 August and Sunday 27 August 2007. The interval length amounts fifteen minutes with a sliding window mechanism of three windows.

6.5.1.1 Malware / Spamer (129.132.153.247)

This host behaved abnormal in two ways. It was infected with the PWS-Banker [52] malware. This back door opens a random port where it listens for remote commands sent by a malicious user. This infection was firstly recognised by the security team on Tuesday morning and secondly on Wednesday morning. This recognition times are depicted with the two red boxes on the left side in Figure 6.19.

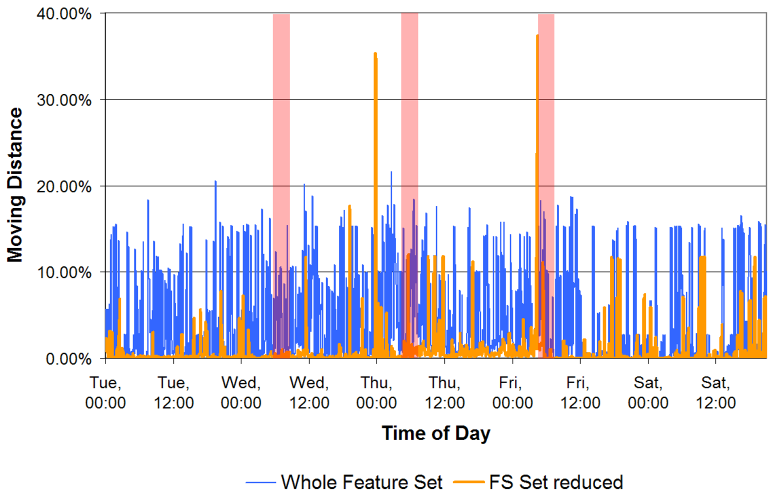


Figure 6.19: Moving Behavior of 129.132.153.247

The diagram shows the distance the host moves among the time intervals. For comparing purposes, we plotted the moving distance by using the whole feature set as well (blue line).

Between this two recognition times, the host moved several times by a big distance. Especially at midnight it changed its position by 35%. This could point toward a possible change in the hosts behavior.

The other anomaly caused by this host was detected at Friday morning. This can be seen by the right most red box in Figure 6.19. The suspicious host has started to send spam mails. Exactly before this detection, the host behavior changed by nearly 40%. The moving distance by using the whole subset gives us much less information, it appears more like Gaussian noise. We see that the reduction of the feature set made the behavior more distinguishable.

On the other hand, between every time interval is a host which moves a long distance and this host is not considered as malicious by the SWITCH team. The average value of the furthest moving distance in each interval is 40% with a standard deviation of 7%. This is even bigger than the maximum moving of our suspicious host. This shows that the moving distance can give a hint but it is not enough to decide only because of that.

When we compare the moving behavior of this malicious host with the average moving distance of all hosts, we see that the malicious host mostly follows exactly this mean value. Even at the points we consider as suspicious, the average moving is as big as for the malicious host. This shows again that the moving distance for itself can not lead to any results in anomaly detection. The average moving distance of all host comparing with the moving distance of the malicious host is showed in Figure 6.20.

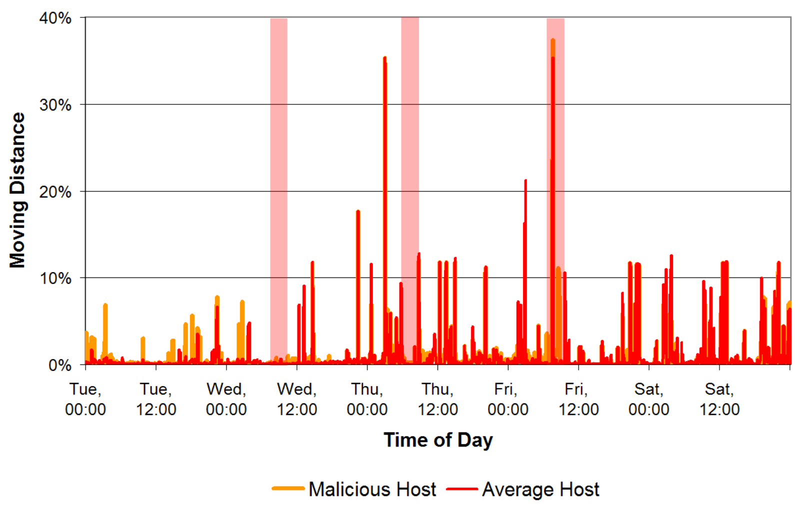


Figure 6.20: Moving Behavior of 129.132.153.247 vs Average of all Hosts

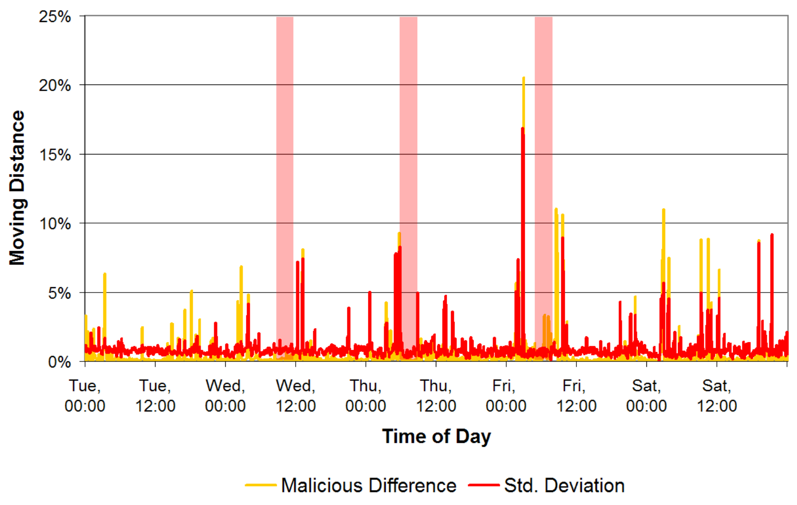


Figure 6.21: Moving Discrepancy from Average Value

But when we calculate the difference in the average moving distance of all hosts and the moving distance of the malicious host we can see some interesting results. Before the first infection was registered, the host behaved different than the average of the hosts. The variance can even be four times the standard deviation. Such a big difference can imply that something abnormal has taken place. On the other hand, the anomaly with the spamming behavior can not clearly be seen. During the spamming process the host does still behave differently than the average host but not significantly. The results can be seen in Figure 6.21. The divergence of the malicious host from the average host is called the Malicious Difference.

6.5.1.2 Botnet (129.132.200.251)

This host could probably be involved in a Botnet[8]. This fact was discovered on Tuesday afternoon as we can see from the red box in Figure 6.22.

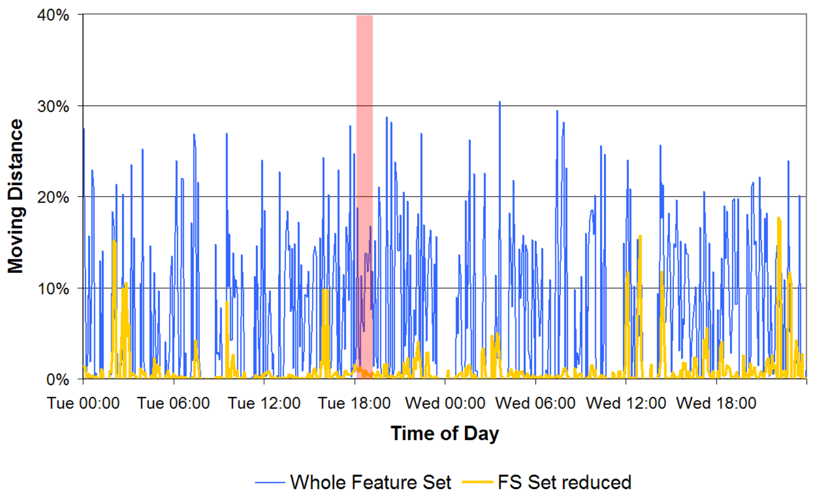


Figure 6.22: Moving Behavior of 129.132.200.251

Again we can see a clear difference between the hopping distances by using the whole feature set or by using the FS Set reduced. With the whole feature set the host changes its position often by more than 20%.

From Figure 6.22 no obvious anomaly can be seen. There are some big changes in the feature space, but never bigger than 20% for the FS Set reduced. Compared with the average movement distance of 40% of the furthest moving host, this is comparable small. In contrast to the abnormal behavior of the malware infected machine, the moving distance of the Botnet host does not correlate with the average moving of all hosts. This circumstance can be seen in Figure 6.23.

We take again a look at the difference in the average moving distance of all hosts and the moving distance of the malicious host (Figure 6.24). Here a possible anomaly can be seen much more clearly. The malicious host moves up to 15% differently in the feature space than the average host. The standard deviation at this point is only 2.5%, so it is six times smaller what looks suspicious.

[8]The computer becomes a bot, because it is remote controlled with many other hosts by a malicious agent.

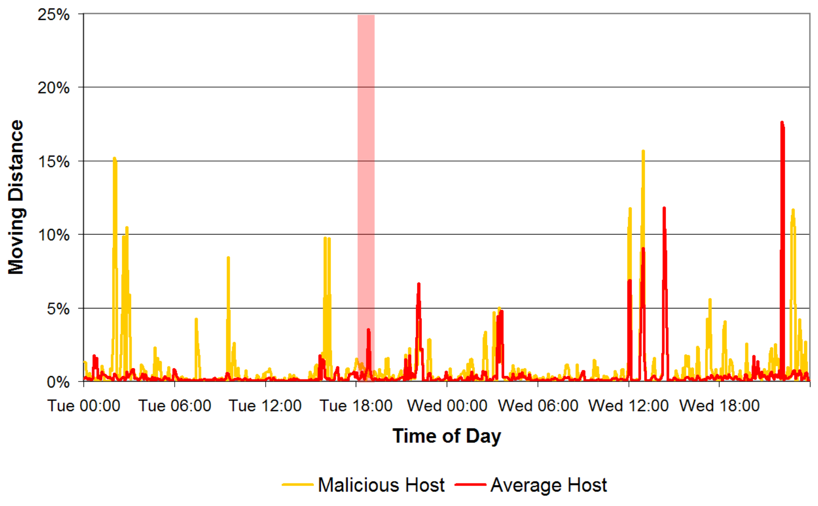


Figure 6.23: Moving Behavior of 129.132.200.251 vs Average of all Hosts

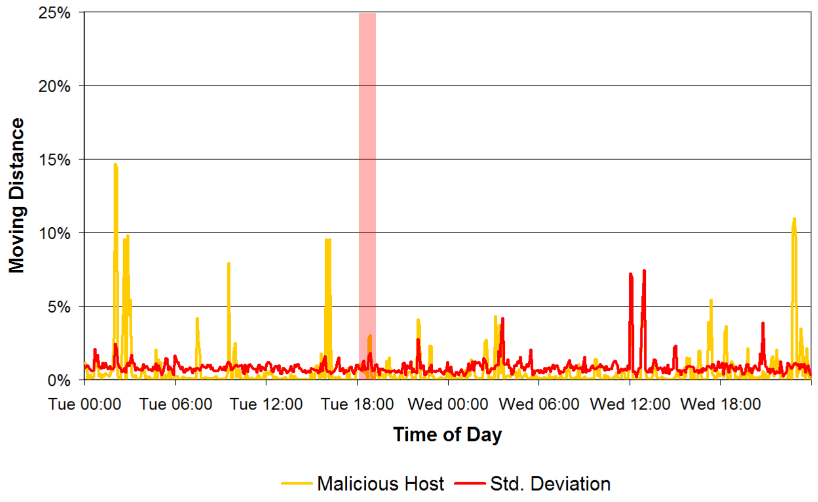


Figure 6.24: Moving Discrepancy from Average Value

6.5.2 Blaster Worm

On August 11th 2003, the W32.Blaster worm appeared. It exploits an RPC[9] vulnerability of Microsoft Windows 2000 and Windows XP, that has been known for some weeks. Characteristic for an infection attempt is a TCP connection to port 135 of the target host. The Blaster outbreak starts around 16:35 UTC, that means 17:35 Swiss time.

We now want to check how such a huge anomaly affects our system.

[9]**R**emote **P**rocedure **C**all

6.5.2.1 Flow Characteristic

By taking a look at the Flow statistic in Figure 6.25 we can see the additional traffic generated by the Blaster worm. The time, when the worm started spreading in the wild is marked with the red background color.

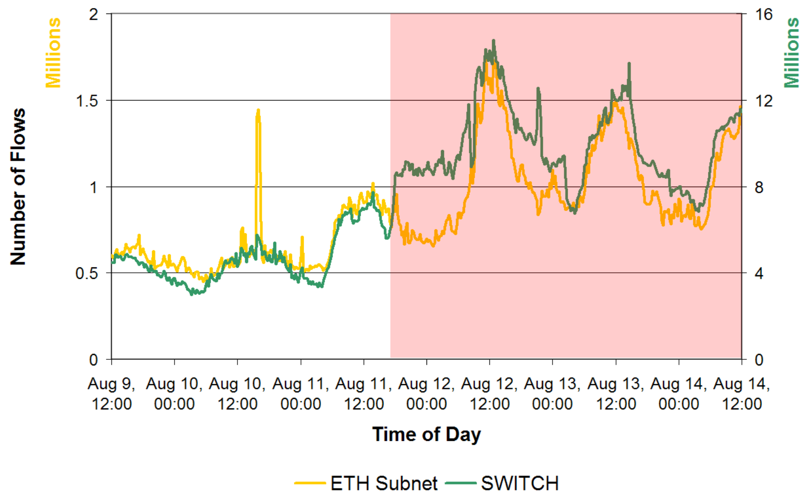


Figure 6.25: Flow Statistic during Blaster Outbreak

We see up to three times more traffic than for a period without any anomaly. This shows the huge network load generated by the infected hosts. It appears that there is a delay in the ETH network of about twelve hours. It takes some time until the ETH subnet gets fully infected.

6.5.2.2 Feature Selection

Correlation Investigating the correlation, we can see no difference between the normal case and the Blaster outbreak. As shown in Figure 6.26, exactly the same number of features are correlated depending on the chosen correlation threshold.

Considering the highly correlated features ($r_{xy} > 0.95$), we see that these are exactly the same features as in the normal case. So the correlation measurement is totally robust against this type of anomaly.

SUD The SUD ranking gets slightly affected by the Blaster outbreak. The Entropy decreases faster than in the normal case. But generally, the behavior is the same. We can see that the characteristic with the fifteen least important features is given as well. More than 80% of this fifteen least important features are the same. So we can say our selected features from the normal case are also good ones one for the abnormal case. Generally speaking, the SUD Ranking is robust against anomalies. The progress of the entropy can be seen in Figure 6.27.

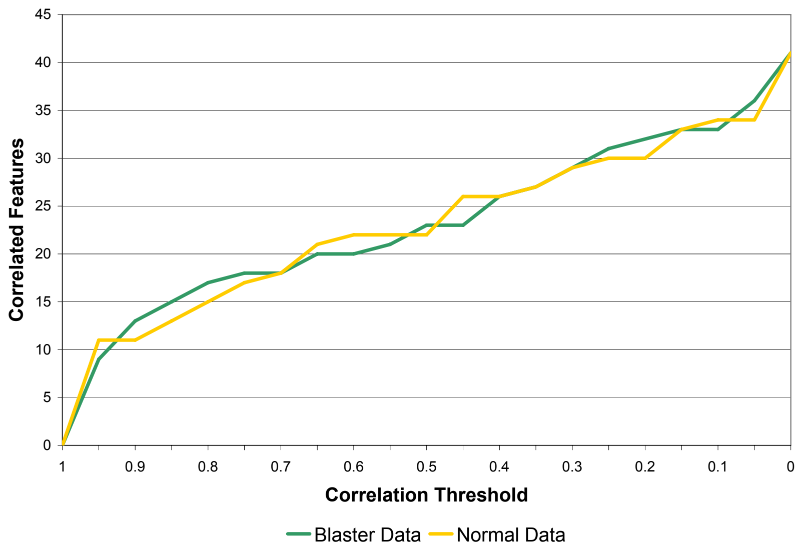


Figure 6.26: Number of correlated Features

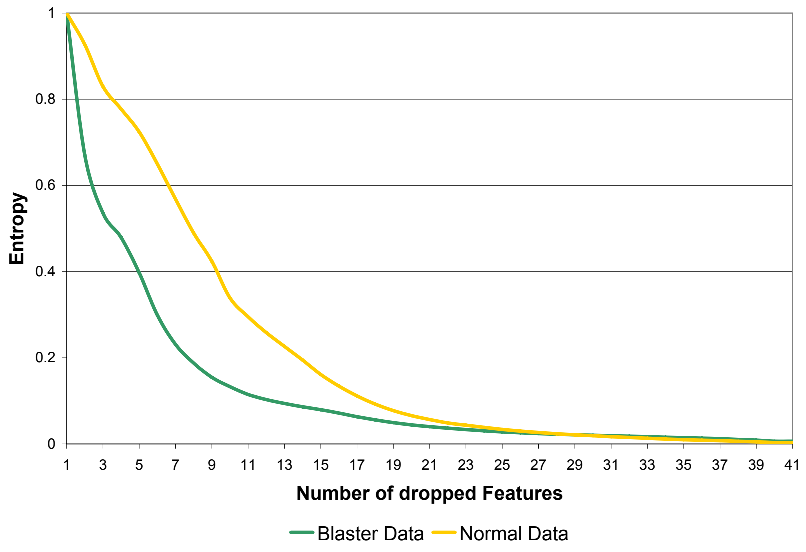


Figure 6.27: Remaining Entropy depending on the Number of dropped Features

FS Algorithm The resulting feature ranking build upon abnormal data looks nearly the same as the ranking for the normal data. When we compare Table 6.13 with Table 6.6, we see that the size of the chosen subset is almost always the same. Furthermore, the chosen features match too.

All this results leads us to the statement, that the feature selection process is completely robust against the Blaster anomaly. This is an important result considering the attributes needed by an anomaly detection system.

Correlation Threshold T_{xy}	Number of Features	
	FS Set	FS Set reduced
0.4	14	8
0.5	16	8
0.6	18	9
0.7	19	9

Table 6.13: Final Feature Ranking of abnormal Data dependent on the Correlation Threshold T_{xy}

6.5.2.3 Clustering

When we consider the statistic of infected ETH hosts in Figure 6.28, we see that the infection of the ETH net reaches its peak nearly one day after the outbreak has started. There are only barely 200 host which are actually infected. This are only 0.3% of the ETH hosts. This signifies that the ETH network is well protected. This could lead to the problem, that findings for this specialised network are not adaptable to a arbitrary network.

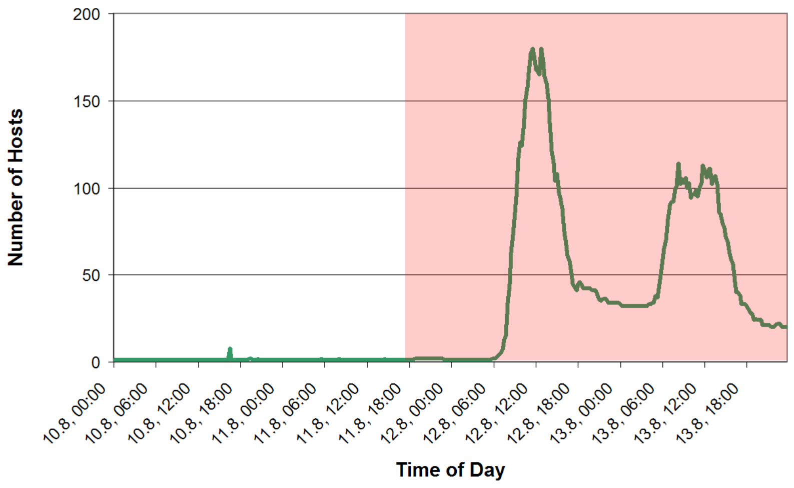


Figure 6.28: Number of infected Hosts in the ETH Subnet

FS Set reduced We clustered some intervals during noon of Tuesday, 12. 08. 2003, when the ETH infection peaked. In a first step we made some clusterings with the FS Set reduced. During this analysis, we made the interesting observation, that O-Cluster only used the feature *min_recv_flow_length* for splitting between clusters. This feature seems to be infected the most in a 'positive' sense by the Blaster worm. Positive means that the heavy-tailed distribution gets flattened.

This could be the case because all hosts receives a lot of connection attempts on TCP port 135 as a result of the Blaster anomaly. For many hosts, this feature is zero, but with the presence of the Blaster worm it increases to a non zero value.

When we compare the feature distribution of the same three features as in Figure 6.9. We see from Figure 6.29 that the data points are more localised around the zero point. Only the feature distribution along the *min_recv_flow_-length* axis had become more widely spread.

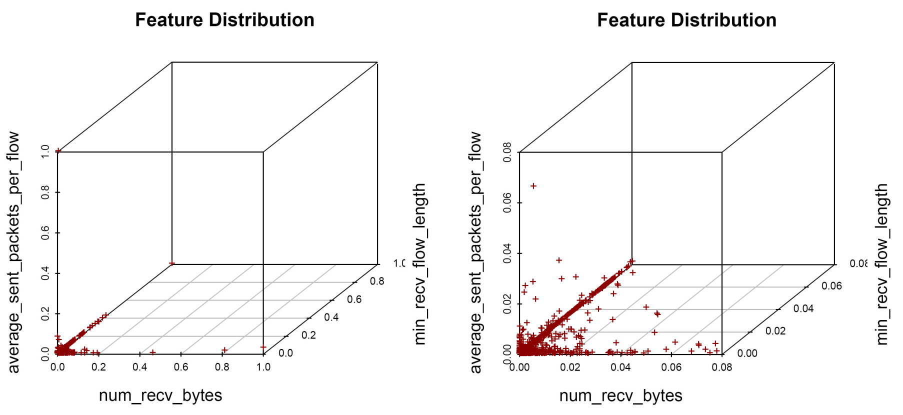


Figure 6.29: Distribution of the Data Points

For a better illustration of the found clusters, we remove the *average_sent_-packets_per_flow* feature and generate a two dimensional plot (Figure 6.30). As said before, all splittings have taken place along the *min_recv_flow_length*-axis. The Bayesian Classifier has again problems by assigning the data points into the different clusters. In contrast, the Direct Rule Classifier has no problem with assigning the data points. A possible problem of the Bayesian Classifier can be the short inter cluster distance.

Whole Feature Set In a second step we made clusterings with all features in order to see how many features are chosen by O-Cluster for clustering. With a sensitivity parameter of $\rho = 1$, the algorithm uses at most nine different features. This is clearly less than the average number of used features for normal data (see Section 6.3.2).

All of this nine features are included in Table 6.11. This shows that the importance of the features stays the same, no feature which has not be relevant for clustering in the normal case will now become relevant in the abnormal case.

Summary The investigation of the cluster did not point out any abnormal behavior. There was no cluster containing all the infected hosts. Again we could see a few big clusters containing all kind of hosts surrounded by some small clusters with a few hosts. The only impact we could monitor was the change in the feature distribution. Most of the features became more heavy-tailed. On the other hand, the *min_recv_flow_length* feature made exactly the

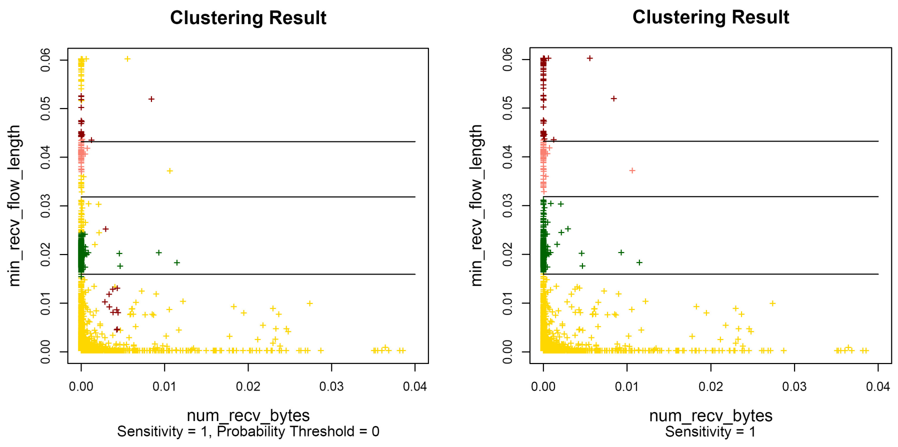


Figure 6.30: Bayesian Classifier vs Direct Rule Classifier

other progression.

6.5.2.4 Profile Movement

The Blaster worm is an anomaly which affects the whole net. For this reason, we expect to see this impact in the average moving distance of the active hosts between two consecutive time intervals. If the behavior of the net changes, than there should be one short peak. After this peak the average moving distance should be low again till the anomaly is over. Then there will be again a peak which signaled the return into the normal behavior.

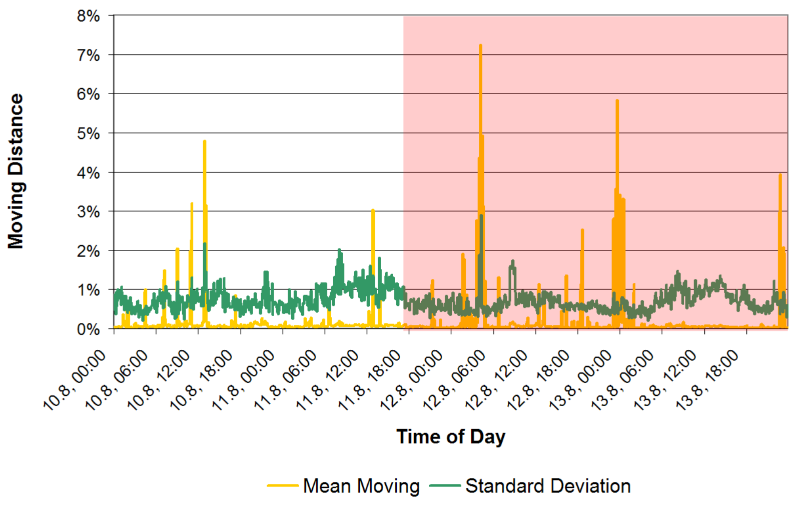


Figure 6.31: Moving Distance of all Hosts and the Standard Deviation

We can see from Figure 6.31 that at the beginning of the Blaster outbreak the ETH net does not change. It takes more than twelve hours till we can see a big peak and all hosts move on average by 7.5% in the feature space. But as we have seen in Figure 6.25 and Figure 6.28, the effect of the Blaster worm on the ETH net was delayed. The peak we can see on Tuesday twelve o'clock does perfectly match to the time when the change in the behavior in the other diagrams took place.

But a change of only 7.5% seems to be a bit small for such a big impact. we can see in Figure 6.31 a peak of 5% before the outbreak started. This is not much smaller than during the outbreak. This shows again that this measurement can only give indications towards possible anomalies but it is never sufficient to make a decision out of it. In the experiments with single abnormal hosts we saw that the average moving of all hosts even can be bigger than this 7.5% although we are confronted with a normal behavior.

Chapter 7

Evaluation

In this Chapter we analyze the results from Chapter 6. The Chapter is again divided into the major steps, as they are Profile Calculation, Feature Selection, Clustering and Profile Movement.

7.1 Profile Calculation

Calculating the profile intervals out of the NetFlow data could be made in an reasonable speed. The calculation time is short enough to deploy it in a real time application. The biggest problem occurs related to the memory usage. For fast execution, a lot of memory is needed. The speed sinks or swims with the available memory. Nevertheless, even if the number of flows increases in the future, the calculation can be done fast enough assumed there is enough of memory available.

The calculation time can be affected by the type of features which have to be calculated. Features like min, max or average values do not influence the calculation time. Because no further information has to be stored and it can be constantly calculated. If we include features like the number of different connected ports, we do need to store information. This storing process needs a lot of memory, but the actual challenge is a fast lookup of this elements. Because with every new value we have to check if it is already stored in the data base. But if such a data base is build for a specific value like different ports, a new feature based on this value does not increase the calculation time any further.

7.2 Feature Selection

We have seen that the feature selection process works quite well. It does not matter what kind of data we choose for feature selection, the resulting features are always the same. The correlation measurement as well as the SUD measurement are robust against any type of anomaly.

This is an essential fact, because the feature selection process is only done once and should deliver the most important features for any kind of data. This means a feature should also be an important one when we process normal data as well as when we are confronted with abnormal data.

The important features stays also the same when we consider profile intervals with different interval length. This is very important, because the interval length has possibly be adapted to the actual circumstances. Especially the correlation measurement is immune against different interval lengths.

Because of the high time complexity of the SUD algorithm, it is viable that a good feature subset can be found even when we do not consider the whole set of active profiles. As we have seen in the Section about SUD (6.2.2), already using 5% of the profiles of all active hosts delivers practical results.

7.3 Clustering

7.3.1 Cluster Interpretation

There was a big problem towards the behavior recognition. It was not possible to separate clusters with different behaviors. Both approaches failed, the Data aspect as well as the Feature aspect.

For the data aspect we have chosen a set of hosts. The behavior of these hosts is known to us. For example the ETH web server, the mail server or the DNS servers. Either we could find any kind of host in a resulting cluster, or there was no such host in the cluster, especially in really small ones. For this reason we have chosen some cluster members and analyzed their behavior in order to investigate the cluster behavior. We have chosen the three hosts which are closest to the cluster center. In some cases it was enough to resolve the IP address in order to extract the purpose of the host out of the hostname. But in most cases the host names were to cryptical to suggest its purposes. These hosts got investigated further by using the network scanner nmap[1] [53]. Nmap allows us to search for offered services on the target machine. But unfortunately, this method did not deliver satisfactory results. Mostly, there was no specific behavior recognisable. Searching hosts which are closest to the cluster center can also be problematic. Because the cluster center has not to lie inside the actual cluster region. In such a case, the behavior of the nearest hosts can be misleading.

The key idea by the feature aspect is the interpretation of the feature values. A big problem arises from the fact, that a lot of features are difficult to interpret. There are features which are easier to interpret than others, like the number of connected IP addresses. But in most cases, the features which are suited for interpretation are dropped during the feature selection process. The resulting features like the average flow length is nearly impossible to interpret in a suitable way. The number of features has also an impact on the interpretability, because if the feature number increases it becomes more difficult to interpret the behavior of a cluster as a whole. Moreover, the value ranges of these features could even be overlapping because of the high dimensionality. This makes the understanding of the feature values even harder.

A reason because of the problem we encountered with clustering could arise because of the distributions of the features. Nearly all features have a heavy-tailed distribution. That means that most objects are concentrated around a typical value, but there are still some data object who can lie far away. Heavy-tailed distributions are much less strewn than normal-tailed distributions.

[1]Nmap stands for Network Mapper

A possible solution to get better clustering result could be choosing different features. If the distribution is less heavy-tailed, it is easier for the O-Cluster algorithm to separate between cluster regions. Nevertheless, this could be a difficult task, because we observed that nearly all generated features are heavy-tailed. NetFlow generally would provide us with more attributes than this nine pointed out in Section 4.1.2. For example the used TCP flags in each flow. Especially such kind of information would be important when investigating security issues and would probably leads to more promising features.

7.3.2 Cluster Evolution

Comparing two clustering results of consecutive profile intervals is a challenging task. Because the number of resulting clusters can vary considerably. This makes it difficult to describe the advancement of each cluster, because it can mostly not be identified in the succeeding time interval. If the sensitivity parameter ρ is decreased, less cluster will get found and so the discrepancy between consecutive intervals decreases. But the number of resulting cluster becomes to small with too many members. That makes it even harder to assign a typical behavior to each cluster.

We compared the cluster between succeeding intervals upon distance measurement. But as we have seen before this is not a good measurement for high dimensional feature spaces. It would be an essential condition to recognise each cluster in the following time interval in order to investigate its behavior and so to detect possible anomalies.

Hosts which belong to the same cluster for one profile interval do not necessarily belong to the same cluster in the succeeding interval. This is mostly the case for small clusters. The fluctuation between is too vast for assigning a static behavior. This big fluctuation can again be explained based on the feature distribution. The cluster lie that close together that a slight change in one feature value can cause a change in the cluster affinity of the host.

7.3.3 Clustering Algorithm

Probably another clustering algorithm can provide better results. The number of available clustering algorithms is enormous but also quite confusing. As stated in Section 2.1.3, a distance based clustering is not well suited for high dimensional feature spaces. For this reason, a lot of algorithms can be neglected. There exist some algorithm which uses totally different approaches like the wavelet transformation in WaveCluster. Problematic is how they perform in high-dimensional environments. Often these algorithms are only tested with low dimensional data.

The feature selection process dropped a lot of our features because of the high correlation. This leads to a comparable low dimensionality. Still high dimensional according to clustering terms, but maybe too low dimensional for O-Cluster to deliver good results. The axis parallel splitting mechanism can not provide good results in comparable low dimensions. For this reason, another clustering algorithm which is suited for high dimensional data could have delivered slightly better results.

7.3.4 Outliers

Finding of 'real' outliers was not possible. The obvious outliers were hosts which were actually no outlier. They had a significant different profile but their behavior was not abnormal. A good example is the host *swisstime.ethz.ch*. For a lot of features it constituted the maximum values. Removing this extreme hosts did not make the feature distribution less heavy-tailed, we could observe for most features a self similar[2] behavior of the distribution. We can not just remove this extreme host, because in an unknown network this could possibly be a real outlier. Nevertheless, such extreme hosts makes it infeasible to search and find some real outliers.

7.3.5 Anomaly Detection

The fact that we could not detect a typical behavior in the normal case makes it even harder to apply the system to the abnormal case. But still we saw that an abnormal behavior affects the feature distribution. Some features became more spread, but the most became more heavy-tailed. Because of that, a promising analysis could include an investigation of the feature distributions and how they evolve during time.

7.3.6 Summary

For all those reasons we were not able to build an reliable anomaly detection system. The clustering delivered not the expected results. We see the main problem in the feature distribution and not with the used clustering algorithm. Such heavy-tailed distributions makes it unfeasible to detect distinct clusters. Choosing another clustering algorithm would not have lead to better results. The only solution is to build better features out of additional information. The features should then always checked towards their distribution.

Additionally, the considered ETH subnet appears as not well suited for our purpose. The network is too specialised to make any statements for an arbitrary network.

7.4 Profile Movement

Because of the disappointing results achieved with clustering, we tried to test the profile intervals with another anomaly analysis method. But as we have seen in Section 6.5 this method can not detect anomalies by itself.

7.4.1 Single Host

Considering single abnormal behaving hosts, we could mostly see an impact on the transfer the hosts made in the feature space. But unfortunately, this impact was sometimes seen in the average moving distance of all hosts as well. This leads to the problem that we can see the impact of a host turning to an abnormal behavior. But if we just see a peak value, we can not say an anomaly has taken place.

[2]Removing the maximum feature value and then plot the feature distribution again with the next lower as the maximum value showed exactly a similar shaped diagram.

Such single abnormal hosts are mostly computers for general purpose use. That means they do not have a typical behavior. If a well known host like the ETH web server is affected by an anomaly, we can expect much better results. Because the behavior of such a host is much more stationary.

7.4.2 The whole Subnet

By considering an anomaly which affects the whole subnet we can again see the time when it happened. But the impact is much smaller than expected. The same eruptions with almost the same peak value can also be seen without the presence of any anomaly. This makes the system unreliable concerning the detection rate of attacks.

7.4.3 Summary

This shows, that this approach is insufficient for detecting anomalies. It can more probably deliver some impulses to the network administrator. Then the administrator has to investigate by itself if something abnormal had happened. But this is not what we are looking for.

Furthermore, this approach uses distance measurements for detecting anomalies. This can cause problems, because as we have seen in Section 2.1.3 that normal distance measurements do not work well in multi dimensional spaces.

Chapter 8

Conclusions

This chapter reports the conclusions we reached as results of this thesis. First, the main findings of the precedent chapters are shortly addressed. The second section contains our recommendations for the next steps in the study of the topics treated in this thesis.

8.1 Summary

Anomaly analysis build upon clustering techniques has previously applied with varying degree of success. The biggest problem is the usually high false positives rate. The detection methods do work well for a typical kind of attack but do not work well for other types. Mostly these detection systems are too specialised.

Our approach was to build a monitoring and detection system which is more various and can detect different attack types. We also applied clustering techniques for achieving this goal.

First we built some profile intervals out of NetFlow data. This task could be solved in consideration of the set goals. We created 41 different features out of the data which describe the behavior of a host.

In the second step of this thesis, several feature selection and clustering algorithms are evaluated and compared against each other. The most promising algorithms have been implemented. The feature selection process did work well, the important features could be found and they were robust against any kind of anomaly.

The results considering the clustering step are disappointing. We could not succeed in extracting meaningful cluster and assign a specific behavior to them. This is a result of the used features, their distribution did not allow to generate well separated clusters.

In a last step we tried to use the profile interval in another monitoring system. As measurement we used the distance a host moves in the feature space between consecutive time intervals. As a result we saw that this system can not lead to a complete detection system. Nevertheless it can produce impulses when something abnormal has taken place.

8.2 Future Work

In this section, we describe which topics we propose to pursue in future.

8.2.1 Profile Calculation

Using totally different features could help to improve the clustering results. It would be desirable to use all attributes NetFlow technically provides. With this attributes a whole field of new features could be build. Especially using the TCP flags could be promising. But unfortunately, many Cisco routers do not export TCP flags yet.

Transformation of the feature space in order to get better shaped distributions. For example a wavelet transformation or similar algorithms. By using such a technique, a new problem arises. The resulting features can not be interpreted anymore.

8.2.2 Feature Selection

The FS Algorithm could be extended. Now it checks in a first step the correlation coefficient and in a second the SUD ranking. This leads to the result, that the final ranking contained some features, which are according to the SUD ranking unimportant. For this reason this features should get filtered out in a additional step too.

8.2.3 Clustering

Choosing a clustering algorithm which is especially suited for heavy tailed data can improve the results. For example the algorithm by Bodyanskiy et al [54] or the algorithm from the paper written by Leski [55]. First of all, these algorithms have to be further investigated. Are they able to process high dimensional data, does the time complexity fit our requirements.

Another approach for anomaly detection can be starting from the outliers. An efficient way of detecting outliers could also lead to an anomaly detection system. Such an approach would probably detect some single abnormal hosts. But it could have problems with an attack who affects the whole network.

Using ODM for clustering is only feasible for experiments. The whole software with its data base is too bloated. For a real world deployment the clustering algorithm should be implemented by ourselves.

Choosing a small subset where all hosts are known can benefit the understanding of the clusters. Moreover, choosing a different subset than the specialised ETH network could lead to better results.

8.2.4 Profile Movement

Considering this moving distance has put out as insufficient. Another measurement than distance metrics would be wishful. For example an investigation of the feature distribution could deliver better results. We have seen that the feature distribution changes during an outbreak. Computing some key features of each distribution and comparing them with consecutive intervals could detect anomalies. Such a system would rather detect some attacks which affects the whole network than single abnormal hosts. Combined with an outlier detection system we could get a comprehensive anomaly detection system.

Appendix A

Raw Feature Table

Nr.	Feature	Description
1	*num_sent_ip*	Number of different host addresses this host has connected to
2	*num_rec_ip*	Number of different host addresses which have connected to this host
3	*num_diff_src_ports*	Number of different source ports the host has connected from
4	*num_diff_dest_ports*	Number of different destination ports the host has connected to
5	*num_diff_incoming_ports*	Number of different ports the host has been connected on by other hosts
6	*num_sent_packets*	Number of packets sent by the host
7	*num_received_packets*	Number of packets received by the host
8	*num_sent_bytes*	Number of bytes the host was the sender of
9	*num_recv_bytes*	Number of bytes received by the host
10	*num_sent_flows*	Number of flows the host was the sender of
11	*num_recv_flows*	Number of flows received by the host
12	*min_sent_flow_length*	Shortest flow the host was the sender of
13	*max_sent_flow_length*	The longest flow sent by the host
14	*average_sent_flow_length*	The average flow length sent by the host
15	*min_recv_flow_length*	The shortest flow received by the host
16	*max_recv_flow_length*	The longest flow received by the host
17	*average_recv_flow_length*	The average flow length received by the host
18	*min_sent_packets_length*	The smallest packet sent by the host
19	*max_sent_packets_length*	The largest packet sent by the host
20	*average_sent_packets_length*	Average packet length sent by the host

continued on next page

Nr.	Feature	Description
21	*min_recv_packets_length*	The smallest packet received by the host
22	*max_recv_packets_length*	The largest packet received by the host
23	*average_recv_packets_length*	Average packet length received by the host
24	*min_sent_packets_per_flow*	Minimal number of packets sent by the host in one flow
25	*max_sent_packets_per_flow*	Maximal number of packets sent by the host in one flow
26	*average_sent_packets_per_flow*	Average number of packets sent by the host in one flow
27	*min_recv_Packets_per_flow*	Minimal number of packets the host has received in one flow
28	*max_recv_Packets_per_flow*	Maximal number of packets the host has received in one flow
29	*average_recv_Packets_per_flow*	Average number of packets the host has received in one flow
30	*num_sent_tcp_flows*	Total number of TCP flows sent by the host
31	*num_recv_tcp_flows*	Total number of TCP flows received by the host
32	*num_sent_udp_flows*	Total number of UDP flows sent by the host
33	*num_recv_udp_flows*	Total number of UDP flows received by the host
34	*num_diff_well_known_dest_ports*	Number of different well known ports the host has connected to
35	*num_diff_well_known_incoming_ports*	Number of different well known ports the host has been contacted on by other hosts
36	*num_diff_not_well_known_dest_ports*	Number of different not well known ports the host has connected to
37	*num_diff_not_well_known_incoming_ports*	Number of different not well known ports the host has been contacted on
38	*most_connected_host*	The computer to which the host has connected the most
39	*most_received_host*	The computer to which the host has connected the most
40	*most_connected_port*	The port number on which the host has the most connected to

continued on next page

Nr.	Feature / Description
41	*most_incomminc_port*
	The port number on which the host has been the most contacted on

Table A.1: Host profile - Overview over all calculated features about a host

Appendix B

Thesis Task

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für
Technische Informatik und
Kommunikationsnetze

Master Thesis
for
Fabian Wanner

Supervisors: Daniela Brauckhoff, Arno Wagner

Issue Date: 02.04.2007
Submission Date: 02.08.2007

Anomaly Analysis using Host-behavior Clustering

1 Introduction

Identifying groups of hosts with similar behavior is very useful for many security applications such as botnet detection, intrusion/extrusion detection, but also for monitoring the services and applications used within a network. Moreover, host behavior clustering is interesting for network traffic modeling since it provides a characterization of the traffic generated by different classes of hosts. In prior work, two unsupervised learning mechanisms (i.e., k-means clustering and an information theoretic approach) have been applied to profile Internet hosts [6, 5].

This thesis aims at comparing different clustering and feature selection mechanisms in order to find an optimal classification mechanism with respect to different constraints such as comprehensibility and stability of the resulting clusters, suitability for exposing anomalous host behavior, and performance issues such as limited computing and storage resources. Following this assessment of different approaches, a design for an optimal solution for monitoring hosts in the ETH campus network will be proposed, implemented, and evaluated.

The data used during this Master thesis is NetFlow [4] data which is collected on the SWITCH (Swiss Education Backbone Network) [2] border routers. This data comprises all aggregated packet headers (flows) which are sent between an ETH-internal host, and hosts which are not part of the SWITCH backbone network. Consequently, ETH-internal traffic and traffic between SWITCH-internal sites is not observed.

2 The Task

This thesis is conducted at ETH Zurich. The task of this Master's Thesis is to find an optimal clustering mechanism for grouping hosts according to their traffic profiles, and to design and implement a system which realizes this optimal clustering mechanism in order to monitor and classify the behavior of hosts in the ETH campus network.

The task of the student is split into three major subtasks that will be: (i) analysis of known mechanisms for clustering and feature selection, (ii) evaluation of two to three clustering and feature selection mechanisms, and (iii) implementation and evaluation of a prototype for monitoring hosts in the ETH campus network.

2.1 Analysis of known mechanisms for feature selection and clustering

Existing mechanisms for feature selection and clustering have to be analysed and compared with respect to, e.g., complexity. Feature or attribute selection is the process of choosing the most relevant attributes for clustering. The student should actively search for and study secondary literature on the problem (unsupervised machine learning, pattern recognition).

2.2 Evaluation of selected mechanisms for clustering and feature extraction

The most promising mechanisms identified in the survey should be evaluated with respect to 1) cluster comprehensibility and stability and 2) suitability for exposing anomaolous host behavior. Candidate features or attributes for clustering are the various pieces of information about a hosts send and receive behavior (e.g., number of flows per minute) that is exposed in the NetFlow data. The student needs to qualitatively assess different feature sets as well as up to three clustering mechanisms. For implementing the clustering approach, we can consider the R [1] statistics tool. Feature extraction should be implemented in C for performance reasons.

2.3 Implementation and evaluation of a prototype

Based on the previous analysis, Fabian will implement and design a prototype for monitoring the host behavior of ETH campus nodes. The design and implementation should take into account that the monitoring framework has to process a huge amount of NetFlow data in real-time on off-the-shelf hardware. Possibly, the UpFrame [3] application can be used to implement the feature extraction step. The prototype must be thoroughly tested under real network load. Therefore, recorded NetFlow traces can be replayed in real-time. A first objective of the testing phase is to provide a proof of concept, i.e. show that the algorithm is correct and that the whole monitoring system is usable.

3 Deliverables

The following results are expected:

- Survey of existing mechanisms for clustering and feature selection.
- Definition of own approach to the problem. In this phase, the student should select the three to four most promising mechanisms. Moreover, the student should provide a design for the monitoring system.
- The specified prototype should be implemented.
- Tests of the prototype with recorded NetFlow data should be made in order to validate the functionality. The implemented machanisms should be evaluated.
- A concise description of the work conducted in this thesis (motivation, related work, own approach, implementation, results and outlook). The survey as well as the description of the prototype and the testing results is part of this main documentation. The abstract of the documentation has to be written in both English and German. The original task description is to be put in the appendix of the documentation. One sample of the documentation needs to be delivered at TIK. The whole documentation, as well as the source code, slides of the talk etc., needs to be archived in a printable, respectively executable version on a CDROM, which is to be attached to the printed documentation.

4 Organizational Aspects

4.1 Documentation and presentation

A documentation that states the steps conducted, lessons learnt, major results and an outlook on future work and unsolved problems has to be written. The code should be documented well enough such that it can be extended by another developer within reasonable time. At the end of the thesis, a presentation will have to be given at TIK that states the core tasks and results of this thesis. If important new research results are found, a paper might be written as an extract of the thesis and submitted to a computer network and security conference.

4.2 Dates

This Master's thesis starts on April 2nd 2007 and is finished on August 2nd 2007. It lasts 6 months in total. At the end of the second week the student has to provide a schedule for the thesis. It will be discussed with the supervisors.

After a month the student should provide a draft of the table of contents (ToC) of the thesis. The ToC suggests that the documentation is written in parallel to the progress of the work.

Two intermediate informal presentations for Prof. Plattner and all supervisors will be scheduled 2 months and 4 months into this thesis.

A final presentation at TIK will be scheduled close to the completion date of the thesis. The presentation consists of a 20 minutes talk and reserves 5 minutes for questions. Informal meetings with the supervisors will be announced an organized on demand.

4.3 Supervisors

Daniela Brauckhoff, brauckhoff@tik.ee.ethz.ch, +41 44 632 70 50, ETZ G93

Arno Wagner, wagner@tik.ee.ethz.ch, +41 1 632 70 04, ETZ G64.1

References

[1] http://www.r-project.org.

[2] SWITCH - The Swiss Education and Reserach Network. http://www.switch.ch/.

[3] Caspar Schlegel. Realtime udp netflow processing framework. Master's thesis, ETH Zurich, 2003.

[4] Cisco Systems Inc. Netflow services and applications - white paper.

[5] Songjie Wei, Jelena Mirkovic, and Ezra Kissel. Profiling and clustering internet hosts. In *2006 International Conference on Data Mining*, 2006.

[6] Kuai Xu, Zhi-Li Zhang, and Supratik Bhattacharyya. Profiling internet backbone traffic: behavior models and applications. In *SIGCOMM '05: Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 169–180, New York, NY, USA, 2005. ACM Press.

09.03.2007

Bibliography

[1] Aapo Hyvärinen, Juha Karhunen, and Erikki Oja. *Independent Component Analysis*. John Wiley & Sons, 2001.

[2] Richard L. Gorsuch. *Factor Analysis*. Lawrence Erlbaum Associates, 1983.

[3] I.T. Jolliffe. *Principal Component Analysis*. Springer, 2002.

[4] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. Automatic subspace clustering of high dimensional data. *Data Min. Knowl. Discov.*, 11(1):5–33, 2005.

[5] Mark Devaney and Ashwin Ram. Efficient feature selection in conceptual clustering. In *ICML '97: Proceedings of the Fourteenth International Conference on Machine Learning*, pages 92–97, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[6] Jennifer G. Dy and Carla E. Brodley. Feature selection for unsupervised learning. *J. Mach. Learn. Res.*, 5:845–889, 2004.

[7] YeongSeog Kim, W. Nick Street, and Filippo Menczer. Feature selection in unsupervised learning via evolutionary search. In *KDD '00: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 365–369, New York, NY, USA, 2000. ACM Press.

[8] Dharmendra Modha and Scott Spangler. Feature weighting in k-means clustering, 2003.

[9] M. Morita, R. Sabourin, F. Bortolozzi, and C. Y. Suen. Unsupervised feature selection using multi-objective genetic algorithms for handwritten word recognition. In *ICDAR '03: Proceedings of the Seventh International Conference on Document Analysis and Recognition*, page 666, Washington, DC, USA, 2003. IEEE Computer Society.

[10] Manoranjan Dash, Kiseok Choi, Peter Scheuermann, and Huan Liu. Feature selection for clustering - a filter solution. In *ICDM '02: Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM'02)*, page 115, Washington, DC, USA, 2002. IEEE Computer Society.

[11] Pabitra Mitra, C. A. Murthy, and Sankar K. Pal. Unsupervised feature selection using feature similarity. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(3):301–312, 2002.

[12] Jose Manuel Pe na, Jose Antonio Lozano, Pedro Larra naga, and Iñaki Inza. Dimensionality reduction in unsupervised learning of conditional gaussian networks. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23(6):590–603, 2001.

[13] Luis Talavera. Feature selection as a preprocessing step for hierarchical clustering. In *ICML '99: Proceedings of the Sixteenth International Conference on Machine Learning*, pages 389–397, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[14] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *In Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining (KDD'96)*, pages 226–231, Menlo Park, CA, 1996. AAAI Press.

[15] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: an efficient data clustering method for very large databases. In *SIGMOD '96: Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, pages 103–114, New York, NY, USA, 1996. ACM Press.

[16] Chien-Yu Chen, Shien-Ching Hwang, and Yen-Jen Oyang. An incremental hierarchical data clustering algorithm based on gravity theory. In *PAKDD '02: Proceedings of the 6th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, pages 237–250, London, UK, 2002. Springer-Verlag.

[17] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. CURE: an efficient clustering algorithm for large databases. pages 73–84, 1998.

[18] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. ROCK: A robust clustering algorithm for categorical attributes. *Information Systems*, 25(5):345–366, 2000.

[19] George Karypis, Eui-Hong (Sam) Han, and Vipin Kumar NEWS. Chameleon: Hierarchical clustering using dynamic modeling. *Computer*, 32(8):68–75, 1999.

[20] Vladimir Estivill-Castro and Ickjai Lee. AMOEBA: Hierarchical clustering based on spatial proximity using Delaunay triangulation. Technical Report 99-05, Callaghan 2308, Australia, 1999.

[21] L. M. LeCam and J. Neyman, editors. *Some Methods for classification and Analysis of Multivariate Observations*, Berkeley, 1967. University of California Press.

[22] Raymond T. Ng and Jiawei Han. CLARANS: A method for clustering objects for spatial data mining. *IEEE Transactions on Knowledge and Data Engineering*, 14(5):1003–1016, 2002.

[23] Zhexue Huang. Extensions to the k-means algorithm for clustering large data sets with categorical values. *Data Mining and Knowledge Discovery*, 2(3):283–304, 1998.

[24] Geoffrey J. McLachlan and Thriyambakam Krishnan. *The EM Algorithm and Extensions*. Wiley-Interscience, 1996.

[25] Mika Sato, Yoshiharu Sato, and Lakhmi C. Jain. *Fuzzy Clustering Models and Applications (Studies in Fuzziness and Soft Computing Vol. 9)*. Physica-Verlag Heidelberg, 1997.

[26] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. OPTICS: ordering points to identify the clustering structure. *SIGMOD Rec.*, 28(2):49–60, 1999.

[27] Alexander Hinneburg and Daniel A. Keim. An efficient approach to clustering in large multimedia databases with noise. In *Knowledge Discovery and Data Mining*, pages 58–65, 1998.

[28] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. pages 94–105, 1998.

[29] Gholamhosein Sheikholeslami, Surojit Chatterjee, and Aidong Zhang. WaveCluster: A multi-resolution clustering approach for very large spatial databases. In *Proc. 24th Int. Conf. Very Large Data Bases, VLDB*, pages 428–439, 24–27 1998.

[30] Wei Wang, Jiong Yang, and Richard R. Muntz. STING: A statistical information grid approach to spatial data mining. In Matthias Jarke, Michael J. Carey, Klaus R. Dittrich, Frederick H. Lochovsky, Pericles Loucopoulos, and Manfred A. Jeusfeld, editors, *Twenty-Third International Conference on Very Large Data Bases*, pages 186–195, Athens, Greece, 1997. Morgan Kaufmann.

[31] Songjie Wei, Jelena Mirkovic, and Ezra Kissel. Profiling and clustering internet hosts. 2006.

[32] Kuai Xu, Zhi-Li Zhang, and Supratik Bhattacharyya. Profiling internet backbone traffic: Behavior models and applications. In *SIGCOMM '05: Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 169–180, New York, NY, USA, 2005. ACM Press.

[33] Nina Taft Thomas Karagiannis, Konstantina Papagiannaki and Michalis Faloutsos. Profiling the end host. 2007.

[34] Thomas Karagiannis, Konstantina Papagiannaki, and Michalis Faloutsos. BLINC: multilevel traffic classification in the dark. In *SIGCOMM '05: Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 229–240, New York, NY, USA, 2005. ACM Press.

[35] Elizabeth Leon, Olfa Nasraoui, and Jonatan Gomez. Anomaly detection based on unsupervised niche clustering with application to network intrusion detection, 2004.

[36] Kalle Burbeck and Simin Nadjm-Tehrani. ADWICE - anomaly detection with real-time incremental clustering. 2004.

[37] Manoranjan Dash and Huan Liu. Feature selection for clustering. In *PADKK '00: Proceedings of the 4th Pacific-Asia Conference on Knowledge Discovery and Data Mining, Current Issues and New Applications*, pages 110–121, London, UK, 2000. Springer-Verlag.

[38] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Michael Wimmer, and Xiaowei Xu. Incremental clustering for mining in a data warehousing environment. In *Proc. 24th Int. Conf. Very Large Data Bases, VLDB*, pages 323–333, 24–27 1998.

[39] Chien-Yu Chen, Yen-Jen Oyang, and Hsueh-Fen Juan. Incremental generation of summarized clustering hierarchy for protein family analysis. *Bioinformatics*, 20(16):2586–2596, 2004.

[40] Boriana L. Milenova and Marcos M. Campos. O-Cluster: Scalable clustering of large high dimensional data sets. In *ICDM '02: Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM'02)*, page 290, Washington, DC, USA, 2002. IEEE Computer Society.

[41] Antonin Guttman. R-trees: a dynamic index structure for spatial searching. In *SIGMOD '84: Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, pages 47–57, New York, NY, USA, 1984. ACM Press.

[42] Yen-Jen Oyang, Chien-Yu Chen, and Tsui-Wei Yang. A study on the hierarchical data clustering algorithm based on gravity theory. In *PKDD '01: Proceedings of the 5th European Conference on Principles of Data Mining and Knowledge Discovery*, pages 350–361, London, UK, 2001. Springer-Verlag.

[43] Alexander Hinneburg and Daniel A. Keim. Optimal grid-clustering: Towards breaking the curse of dimensionality in high-dimensional clustering. In *VLDB '99: Proceedings of the 25th International Conference on Very Large Data Bases*, pages 506–517, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[44] Cisco Systems Inc. *Introduction to Cisco IOS NetFlow - A Technical Overview.*

[45] SWITCH - The Swiss Education & Research Network. http://www.switch.ch/.

[46] Cisco Systems Inc.
http://www.cisco.com/.

[47] M. Dash, H. Liu, and J. Yao. Dimensionality reduction of unsupervised data. In *Tools with Artificial Intelligence, 1997. Proceedings., Ninth IEEE International Conference on*, number 3-8, pages 532–539, November 1997.

[48] Oracle Data Mining.
http://www.oracle.com/technology/products/bi/odm/index.html.

[49] Hierarchical Clustering Explorer.
http://www.cs.umd.edu/hcil/hce/.

[50] R - The R Project for Statistical Computing. http://www.r-project.org/.

[51] CERT: Security Advisory: MS.Blaster (CA-2003-20). http://www.cert.org/advisories/CA-2003-20.html.

[52] PWS-Banker, Malware. http://uk.trendmicro-europe.com/enterprise/vinfo/.

[53] Nmap - Network Mapper. http://insecure.org/nmap/.

[54] Yevgeniy Bodyanskiy, Illya Kokshenev, Yevgen Gorshkov, and Vitaliy Kolodyazhniy. Outlier Resistant Recursive Fuzzy Clustering Algorithms. In *Computational Intelligence, Theory and Applications*, pages 647–652. Springer Berlin Heidelberg, 2006.

[55] Jacek Leski. An ε-Intensive Approach to Fuzzy Clustering. *Int. J. Appl. Math. Comput. Sci.*, pages 993–1007, 2001.