Iterative Principal Component Analysis (IPCA) for Network Anomaly Detection

by

Athanasios Delimargas, **Diploma in Electronics and Computer Engineering**

A thesis submitted to the
Faculty of Graduate and Postdoctoral Affairs
in partial fulfillment of the requirements for the degree of

Master of Applied Science in Computer Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering
Department of Systems and Computer Engineering
Carleton University
Ottawa, Ontario
September, 2015

The undersigned hereby recommends to the

Faculty of Graduate and Postdoctoral Affairs

acceptance of the thesis

Iterative Principal Component Analysis (IPCA) for Network Anomaly Detection

submitted by **Athanasios Delimargas**, **Diploma in Electronics and Computer Engineering**

in partial fulfillment of the requirements for the degree of

Master of Applied Science in Computer Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering

Department of Systems and Computer Engineering

Carleton University

September, 2015

Abstract

Due to a plethora of network anomalies and their rapid increase both in complexity and diversity, an ongoing research for security countermeasures is emerging. Principal Component Analysis (PCA) is one of the several methods that were suggested in order to detect such anomalies and is known as a powerful tool in finding and diagnosing anomalies in network traffic. Nonetheless, previous relevant research work highlighted some inconsistencies of the classical method. It has been shown that the efficiency of the results are highly dependent to the input data and the calibration of its parameters. These parameters should be carefully selected in order to pinpoint the existence of anomalies in network traffic. By obtaining real network traffic traces from a small enterprise and artificially injecting anomalies, we apply a modified PCA based method. The results of our experimentation imply that this method possesses promising capabilities in efficiently detecting network anomalies and manages to surpass some of the limitations of the classic approach.

I dedicate my thesis work to my family and my friends. A special feeling of gratitude to my parents, Kostas and Athina Delimargas whose words of encouragement were vital to the completion of this work.

Acknowledgments

This research was partially supported by the Defense Research and Development Canada and Solana Networks. I am thankful for this opportunity, the insight and expertise they provided that greatly assisted the research.

I would also like to acknowledge all the people who contributed, each in his/her own way, towards the accomplishment of my thesis. First of all, I would like to thank my supervisor, Ioannis Lambadaris, for his continuous guidance and his endless patience. In addition, my research fellows, Emmanouil Skevakis and Hassan Halabian for sharing their thoughts and ideas during the completion of this research work.

Last but not least, I would like to thank my family for their continuous love and support.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In the past decades, the Internet has provided the means for worldwide interconnection and communication. As the basis of e-commerce infrastructure, its impact n the worldwide economy is critical. From small to large enterprises, universities, institutes and households, the variety and number of connections to the Internet are growing exponentially. Unavoidably, this has also resulted in an increase in both the complexity and diversity of cyber threats and attacks. From this perspective, network security and anomaly detection are evolving into significant research areas. In previous decades, malware and denial-of-service attacks (DoS) mainly resulted in network downtime costs and compromised system recovery time. In the recent years, more sophisticated and complex types of attacks have continuously been emerging. Attacker's motives these days include industrial espionage, system sabotage and extortion, to name just a few. For commercial organizations and governments in particular, a network attack could negatively impact their reputation and cause significant financial loss. A steady demand for network security tools is emerging in order to protect critical infrastructure, client information or even corporate secrets.

1.1 Thesis organization

The structure of this thesis is structured as follows. In this chapter we provide a brief introduction to the network anomaly background, our problem statement, research contributions and our research methodology. In Chapter 2, we review the relevant research literature. In Chapter 3, we introduce the Principal Component Analysis (PCA) approach and provide justification for our iterative PCA (IPCA) approach. In Chapter 4, we provide details about our data trace and the anomalies that we inject utilizing the FLAME tool. In Chapter 5, we discuss the experimental procedure and describe our experimental framework. In addition, we provide details about the artificial injection of the network traffic anomalies and their role in evaluating the anomaly detection capabilities of our algorithm. In Chapter 6, first we examine and discuss the behavior of IPCA when there are no artificial anomalies injected into them. We then discuss about the different types of experiments that we run after the artificial anomaly injection and derive valuable conclusions from the results. In addition, we discuss the feature modifications that we applied in order to increase our anomaly detection rates and we show that IPCA performs better than the classical PCA in most of the cases. Lastly, in Chapter 7, we draw conclusions about the methodology and the results and provide proposals for future research.

1.2 Network anomaly background

In an abstract form, a network traffic anomaly is defined as any network activity or phenomenon that results in abnormal network traffic patterns. This change in traffic patterns could be due to traffic outages, flash crowds, misconfigurations, vendor implementation bugs, network attacks, network worms and so forth [1]. The above definition implies that such an anomaly can be either malicious or benign. In either case, these suspicious events should be expediently brought to the attention of the network operator. In addition, reliable information should be provided that will allow to decide if there is a security problem that necessitates the deployment of security countermeasures. For example, if the anomaly is a security problem in the network (e.g.a DoS attack), the network administrator would be able to detect it quickly and then block the necessary IP addresses and disable the relevant ports. If the anomaly is instead related to a traffic outage or misconfiguration, the operator can act quickly and fix the problem before missing more traffic.

In general, there are three main tasks carried out by a good network anomaly detector : *i)* detecting with precision the time when a suspicious event is taking place, *ii)* identifying what type of anomaly is taking place and *iii)* quantizing an anomaly and determining its importance or the hazard it is causing to the network. This thesis focuses on the detection task of anomaly detectors, specifically, algorithms that can efficiently pinpoint the occurrence of network anomalies at a given point in time.

There are two general types of network anomaly detection [33]: network based and host based. The latter focuses on constantly monitoring an end host (i.e. the personal computer of an enterprise employee) while the former focuses on monitoring the way in which network information flows in our network. Our approach focuses on network based anomaly detection.

1.3 Problem statement and research contribution

Our primary research goal is to gather information about the traffic of the two main network protocols (namely TCP and UDP) and to aggregate this information into a single traffic measurement matrix. We deploy simple metrics such as the number of bytes, packets and flows with entropy features such as the entropy of the source and destination IP addresses and ports in order to develop an algorithm that takes both types of features into consideration before it decides if an anomaly actually exists. We test the efficiency of our algorithm via numerous experiments. In addition we provide modifications of the classical PCA approach in order to achieve higher detection rates. We also provide reasoning for both types of features that we include in our experiments and those that we decide to leave aside. In addition, by modifying the features and experimenting with different sets of features, we explore their efficiency in detecting network attacks while keeping the computational complexity at a relative low level. Moreover, we see why the classical approach performs rather poorly in certain cases and how our approach is improving the anomaly detection rates.

In our work, we focus on unsupervised learning. In general, there are many statistical approaches for unsupervised network anomaly detection, including covariance matrix analysis, PCA and wavelet analysis techniques as well as a combination thereof. In our work, we focus on PCA cite[3,4]. We provide details about this approach and propose certain modification in order to overcome its inconsistencies. The contributions of our work are as follows:

1. We show that our approach always has the same or better results in comparison to the classical approach with an acceptable trade-off in computational complexity.

2. We diminish the need for careful calibration of the parameters in order to properly define PCAs normal and abnormal subspaces. In our approach, only one parameter (the iteration stoppage point) needs to be defined. This plays an important role in creating a more automated and general approach.

3. Large volume anomalies capture a lot of variance and become part of the normal subspace. This is one of the primary problems of the classical approach, namely

the normal subspace poisoning. Our approach shows that it can overcome this limitation in most of the cases.

4. We implement modifications to the traffic features in order to improve our algorithm's anomaly detection performance.

1.4 Research methodology

In this research thesis, we focus on applying a low complexity algorithm in network traffic traces in order to detect anomalies in an unsupervised manner. To be more precise, we explore the potentials of a modification of a statistical procedure that is known as the Principal Component Analysis (PCA). In experimenting with the classical PCA method, we meet with most of the known inconsistencies and limitations that are described in the relevant literature [1]. We attack these problems by proposing a modified iterative approach. We then experiment with different traffic features in order to define the most appropriate set, in terms of higher anomaly detection rates. Our goal is to combine information from simple measures of the network traffic such as bytes, packets and flows in conjunction with entropy features such as IP source and destination addresses and ports. Although the experimentation is conducted off-line, our approach could be viewed as the basis for a low complexity algorithm that will be able to detect traffic anomalies in real time and will overcome some of the limitations of the standard PCA approach. Our approach should not be seen as an automated algorithm for anomaly detection but rather as a tool for aiding the network operator in identifying potential network anomalies in the two main network protocols of the Internet Protocol Suite, namely Transmission Control Protocol (TCP) and User Datagram Protocol (UDP).

By gathering real network traffic traces and injecting artificial anomalies, we are able to test the efficiency of our algorithm and define the conditions under which anomaly detection is feasible. In addition, we show the advantages of our modified approach as compared to the classical PCA approach. Contrary to most of the relevant research, our network traffic datasets are daily network traffic traces from small enterprise networks rather than weekly traffic traces from backbone networks. In addition, most of the relevant research in the literature where is conducted in time bins that vary from ten minutes to day-long. In our work, we construct very small, 1.25 minute time bins. These very small time scales play an important role in enabling us to detect anomalies in real time in the future.

Chapter 2

Literature review

Network anomaly detectors are classified into two main categories. The first category is supervised learning [2,3] which includes algorithms that require training data in order to construct a model of the network behavior. These labeled datasets are needed for training purposes. In addition, these datasets should be carefully selected in order to reflect a proper-general representation of the traffic in a network. If this is not done, the results will be rather poor. In general, supervised learning requires constant updates so as not to become obscure and shows an inherent limitations in detecting new attacks. However, its results lead to low false positive rates since the algorithm is trained to detect known attacks. Hence, these algorithms define anomalies based on the deviations from the derived model and the statistical information of the system's history (i.e. training data). There should be sufficient training samples in order to model both normal and abnormal activity. Based on this information, the algorithm makes a decision on a new traffic trace. It is clear that the training samples should be of the same class as the samples investigated. If this is not the case, the results will be miscellaneous. This imposes a hard boundary on the generality of the algorithm. In addition, zero day attacks would be almost impossible to detect since the algorithm would not be trained to detect them.

In contrast, the second category of network anomaly detectors includes algorithms that do not require training data [4,5,6,7]. This is the unsupervised learning. They act, based on information concerning the current state of the network and do not incorporate any prior knowledge to the model. On the one hand, unsupervised learning removes the limitations of supervised learning. It is more general and does not require labeled datasets. Moreover, these algorithms perform better than the supervised algorithms when faced with unknown or zero day attacks. On the other hand, they could result in higher false positive rates due to their generality [8].

2.1 Related work

A review of the relevant research reveals several approaches on network anomaly detection [1]. Shelulin et al [9] discuss the discrete wavelet decomposition of the traffic in conjuction with F-test algorithms of statistical processing. The authors show that wavelet analysis, in different levels of decomposition, can achieve high network anomaly detection rates. Han Li et al [10] use cluster analysis (k-means) to detect abnormal network traffic patterns. Ahmend et al [11], propose an online anomaly detection based on kernel recursive least squares. This technique is similar but not identical to PCA; it can achieve both lower computational complexity and faster detection time than PCA. The main idea is that since network traffic has low dimensionality, it can be represented by a small set of linearly independent feature vectors.

Lahkina et al [5] pioneering work popularized PCA anomaly detection in the network community. In this study, the authors suggest a general approach in network wide traffic anomaly detection. Specifically, they introduce the normal and abnormal subspace construction method by means of PCA. Their initial work is applied only to volume traffic. Later, in [6], the authors expand their work by adding IP addresses and port entropy features. By gathering volume data from different links of the network, they show that they are able to detect network-wide traffic anomalies.

Kwitt et al [12] focus on PCA unsupervised network anomaly detection. They show that a shift from supervised to unsupervised anomaly detection is necessary in order to achieve more robust network anomaly detection results. The authors argue that the utilization of the Minimum Covariance Determinant (MCD) estimators could potentially play an important role towards this goal. Callegari et all [13] introduce the Kullback Leibler divergence in order to achieve improved detection rates when compared to the classical PCA approach. The authors propose a method for identifying network anomalous flows in an aggregated manner.

Brauhoff et al [4,14] focus on PCAs sensitivity to its input parameters. Morevoer, the authors argue that another PCA drawback is that it does not consider the temporal correlation of the variables. To overcome this obstacle, they suggest the Karhunen Loeve expansion as the appropriate framework and provide a Galerkin method to build their predictive model. Jiang et al [15] focus on both detecting and localizing an anomaly in network data streams. By extending their sparse PCA framework with multidimensional Karhunen Loeve expansion, they show that their approach provides promising network anomaly localization capabilities.

Huang et al [16], use distributed tracking and PCA analysis and propose an online network anomaly detection architecture. In this work, the authors develop a trade-off quantization between overall anomaly detection accuracy and limiting the size of the data that are transmitted to the central coordinator/detector. Their approach also utilizes PCA to Border Gateway Protocol (BGP) updates in order to detect link and node failures.

Important work conducted by Ringberg et al [17] shows that it is difficult to effectively tune the parameters of a PCA anomaly detector. Specifically, they show that PCA is very sensitive to the number of eigenvectors that comprise the normal subspace. In addition, a large anomaly may contaminate the normal subspace, causing detection to be impossible. As we see in Chapter 6, our proposed method attacks both of these problems.

Yang Liu et al [18] propose a sketch based streaming PCA algorithm that focuses on PCAs scalability issues. Their algorithm runs in logarithmic time in contrast to the linear running time of PCAs classical approach. In one of the most recent works, Duo Liu et al [19] use the Machalanobis distance function, the temporal effect of the network traffic data and the entropy features of the traffic like source and destination IP entropy, in order to construct a more robust PCA approach than the standard PCA. Kudo et al [20] attack the normal subspace poisoning. This happens when excessively large anomalies contaminate the normal subspace resulting in poor anomaly detection performance. The authors exploit the daily or weekly periodicity of network traffic and construct a network traffic model of the normal behavior. Their work focuses on large backbone networks and specifically the Abilene network.

The work of Zhang et al [21] focuses on the problem of normal subspace poisoning. The authors work shows that a few strong Origin-Destination flows can severely contaminate the normal subspace and cause large anomalies to be undetectable. Duong et al [22] propose a semi-supervised PCA approach. They use a modified Machalanobis distance and a K-means clustering method to build a normal profile traffic based on small training datasets. Their work is quite novel since they try to incorporate the advantages of both supervised and unsupervised methods into a middle ground solution.

Zhang et al [23] utilize PCA in a semi-centralized architecture for privacy preserving purposes. Their work focus on large backbone networks and its the first work that tests PCAs capabilities in network traffic detection when privacy concerns are put into perspective. They show that PCA can effectively detect traffic anomalies without forcing ISPs to reveal private traffic information. Li et al [24] propose a method to detect DoS attack by exploiting the spatial correlation between attack flows. They utilize the Hurst parameter in conjunction with the variance and auto-correlation of the DoS attacks flows. Their algorithm detect attacks my exploiting the strong correlation between the anomalous flows of DoS attacks and is more scalable than the network wide anomaly traffic.

In wireless networks, Livani et al [25] present PCACID, a PCA-based centralized approach and PCADID, a PCA-based decentralized approach, both for intrusion detection. PCA is used from every monitor node of the wireless network in order to define the normal subprofiles of every node and a global normal profile based on the subprofiles of the other nodes. The authors discuss about the tradeoffs between the distributed and centralizes approaches in terms of memory, energy consuption and performance. Issariyapat et al [26] applied PCA in IP network anomaly detection and not in large backbone networks. They show that PCA is an excellent candidate for the first level of network traffic defense (i.e. a preliminary anomaly detection system). Androulidakis et al [27] focus on the impact of selective sampling in the small flows that are the root of many traffic anomalies. By utilizing real university network date and applying PCA, the authors discuss the tradeoff between detection accuracy and the reduction of the volume of collected data.

Pascoal et al [7] upgrade the classical PCA approach by proposing a robust PCA based on robust statistics. The authors argue that their feature selection algorithm based on mutual information metrics manages to minimize PCAs sensitivity to outliers. Their work achieves very high detection rates in different test scenarios and traffic conditions. Hayang Kim et al [28] focus on scalability issues in large scale networks. They apply Higher-Order Singular Value Decomposition (HOSVD) and Higher-Order Orthogonal Iteration (HOOI) algorithms in order to detect network wide traffic anomalies. Their work outperforms the standard two dimensional matrix SVD/PCA in terms of both false alarm and miss probabilities rates while reducing the algorithmic complexity. Hakami et al [29] utilized PCA in wireless mesh networks mainly for the anomaly identification phase. They showed that PCA can effectively separate the false alarms from the real network anomalies in real time and identify the flows that cause the anomaly. Novakov et al [30] suggest a hydrid PCA approach that combines the PCA statistical analysis with a spectral analysis technique based on Haar wavelets.

Chapter 3

Principal Component Analysis (PCA) and its extension (IPCA)

In the majority of the cases, network behavior analysis and anomaly detection are employed in deterministic, signature based approaches. Nevertheless, there is an evolving interest in the statistical analysis of network traffic. This analysis can tackle numerous problems such as effectively confronting unknown attacks.

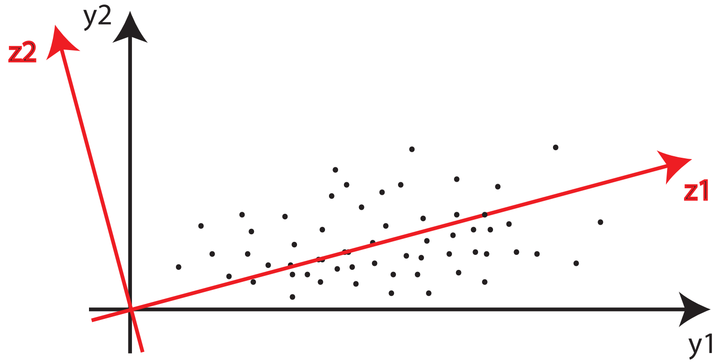

Figure 3.1: Two dimensional PCA

Statistical network traffic analysis is a challenging area of research, especially in large scale networks (due to the great volume of data). Dimension reduction algorithms and particularly the PCA plays an important role in establishing network anomaly detection.

The PCA can be easily understood from a simple example. For illustration purposes, assume data scattered in the two dimensional plane as shown in Figure 3.1. PCA computes a new set of coordinates ($z1$,$z2$) such that the spread around the $z1$ axis is minimized. This enables the data to be represented accurately enough from only one dimension (i.e. $z1$). In this chapter, we discuss the PCA and provide reasoning for our modification. In our study, the input data are a two-hour network

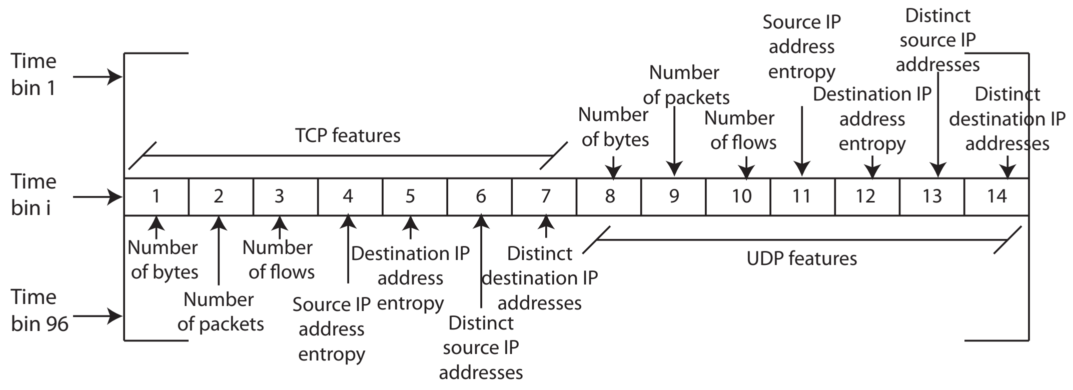
Time bin 1
Time bin i
Time bin 96
TCP features
UDP features
1
2
3
4
5
6
7
8
9
10
11
12
13
14
Number of bytes
Number of packets
Number of flows
Source IP address entropy
Destination IP address entropy
Distinct source IP addresses
Distinct destination IP addresses
Number of bytes
Number of packets
Number of flows
Source IP address entropy
Destination IP address entropy
Distinct source IP addresses
Distinct destination IP addresses

Figure 3.2: List of 14 traffic features collected in one time bin (out of 96)

traffic trace. It is organized in an $m \times n$ matrix, where m is the number of network traffic features and n is the number of time bins (time intervals) that we divide the two-hour traffic trace. These features are: i) number of packets, ii) number of bytes, iii) number of flows, iv) distinct source IP address, v) distinct destination IP address, vi) source IP address entropy and vii) destination IP address entropy. These same seven traffic features are obtained for both TCP and UDP traffic as shown in Figure 3.2. The IP address entropy (source or destination), for a given time bin, is defined as

$$H(x) = -\sum_i p_i \log_2 p_i \ ,$$

where i is a *distinct* IP address in time bin x and p_i is the relative frequency of appearance of the i^{th} address in this time bin.

3.1 Principal component analysis

A given dataset (matrix $Y_{m \times n}$, in our case $m = 14$ and $n = 96$) can be mapped, under linear transformation, into a new dataset $Z_{m \times n}$, with row vectors z_j, as follows:

$$z_j = \sum_{i=1}^{m} w_{ji} y_i, \qquad j = 1, ..., m$$

where y_i is a row vector representing the i^{th} row of the original dataset, z_j represents the j^{th} row of the new dataset and $w_j = [w_{j1}, .., w_{jm}]^T$ is a vector of scalar weights. A necessary condition for PCA is that each row of Y should be of zero mean. This is particularly important because relevant research has shown that if we do not subtract the mean of every feature, our results will be heavily impacted and skewed [6]. The objective of PCA is to find a new coordinate system such that the first coordinate captures the largest variance of the transformed dataset, the second captures the second largest and so forth. Our objective is thus to find a vector w_1 such that the variance of

$$z_1 = w_1^T Y$$

is maximized. In other words, we are looking for the coordinate that captures the maximum energy in the data signal. The variance of z_1 is

$$var(z_1) = w_1^T C_Y w_1$$

where

$$C_Y = \mathbb{E}[(Y - \mathbb{E}[Y])(Y - \mathbb{E}[Y])^T]$$

is the covariance matrix of Y. We are interested in finding w_1 that maximizes the quantity

$$w_1^T C_Y w_1$$

with the constraint that w_1 must have unit length.

In order to solve this problem, we apply the Lagrange multiplier method to maximize the quantity

$$L = w_1^T C_Y w_1 - \lambda(w_1^T w_1 - 1)$$

with respect to w_1. By differentiating L with respect to w_1 and setting the derivative equal to zero:

$$C_Y w_1 = \lambda w_1$$

Therefore, λ is an eigenvalue of matrix C_Y and w_1 is the corresponding eigenvector. By multiplying both sides with w_1^T,

$$var(z_1) = \lambda$$

Hence, by selecting w_1 as the eigenvector that corresponds to the largest value of λ, $var(z_1)$ will be maximized. The axes of this new system are called principal components. For the above to hold, a necessary condition is that the random variables (the rows of Y) are of zero mean.

It can be shown that w_1 is the eigenvector (of unity magnitude) of C_Y, corresponding to the largest eigenvalue of this matrix. The direction for the second largest variance is defined by the normalized eigenvector corresponding to the second largest eigenvalue and so on.

After mapping the data to the new coordinate system, we need to select the number (k) of dimensions that are a "good" representation of our dataset, (i.e. those that capture “most” of the variability. These dimensions will constitute the normal subspace and the rest the abnormal one. However, sufficiently large anomalies can result in significant variance and they are thus included in the first k components [6,17]. In this case, the anomaly becomes part of the normal subspace and is undetectable. The selection of k can be based on the percentage of the total variability in the normal subspace, or it can be a constant number.

3.2 Anomaly detection

As [5] suggests, selecting a fixed number of dimensions as the normal subspace can result in poor performance, due to the variability of different input traces. To overcome this obstacle, we introduce a modified PCA approach. The selection of the principal components defines the normal and abnormal subspaces. We can perform the projection of the 14 traffic features onto the subspaces for every time bin . Let P denote the matrix $14 \times k$ whose columns are the eigenvectors p_i (14×1 vector) that constitute the normal subspace. Then,

$$Y_n = pp^T Y$$

is the projection of the traffic features onto the normal subspace and

$$Y_{ab} = (I - pp^T)Y$$

is the projection of the traffic features onto the abnormal space. An anomaly occurrence will result in a significant change to Y_{ab}. In order to detect such changes, we define the squared prediction error (SPE):

$$SPE = ||Y_{ab}||^2 = ||(I - pp^T)Y||^2$$

In the classical approach, an important decision again needs to be taken; the appropriate SPE threshold must be defined. If the SPE threshold is low, it results in a large number of false positives. If the SPE threshold is high, then the algorithm will not detect anomalies of low intensity. In other words, a very delicate calibration is essential. We tackle this problem, as we see in section 3.3.

3.3 Iterative PCA method (IPCA)

We introduce the IPCA towards the goal of implementing an improved, more robust PCA varitation that which will have minimum dependency on its parameters, such as the selection of the principal components that constitute the normal space and define the SPE threshold. The two main characteristics that differentiate it from the classical PCA approach are as follows:

1. Neither a percentage threshold nor a fixed number of eigenvectors needs to be defined, in order to determine the two subspaces. At each iteration, only the first eigenvector that corresponds to the largest eigenvalue constitutes the normal subspace.

2. No SPE threshold is needed for the detection process. At each iteration, only the time bin with the largest SPE score (regardless of its value) will be denoted as possibly anomalous.

A description of the IPCA algorithm follows:

Step 1 : A 14×96 matrix (denoted as Y) is created from the data trace. We then subtract from each traffic feature (every row of Y) its mean value, thus creating a new 14×96 matrix, B, where

$$B = Y - \mathbb{E}[Y].$$

The input features in different rows (i.e. number of bytes and entropy) are not of the same scale. For example the scale of bytes number is 10^9 and the packets number is 10^6 while entropy features hardly reach 10^1. Moreover, volume traffic features are much more variable than the entropy features. They can therefore get most of the variability of the input data matrix. This problem can be eliminated by normalizing matrix B. However, PCA is sensitive to scaling. In other words, different normalization factors may result in different input matrices and therefore different final results. Motivated by the above, we proceed to the next step.

Step 2 : Divide each row by its standard deviation, creating the matrix

$$B_n = \frac{B}{std(B)}.$$

This is equivalent to applying PCA over the correlation matrix instead of the covariance matrix that is implemented in the classical approach. By doing so, we compress large quantities and stretch small values. In short, the differences between the nature and scale of different features make the conventional implementation of PCA inappropriate.

Step 3 : Construct the correlation matrix of B_n and derive the eigenvalues and eigenvectors.

Step 4 : Choose the eigenvector that corresponds to the largest eigenvalue. This eigenvector constitutes the normal subspace and all of the rest the abnormal subspace. Due to the iterative approach, the need to select more than one eigenvector for the normal subspace is reduced. A new normal subspace will be constructed in each iteration, free from the effects of the anomalous time bin (see Step 6).

Step 5 : Calculate the SPE score of every time bin. The time bin with the largest SPE score is denoted as possibly anomalous.

Step 6 : Remove the effects of this time bin by setting each features of this time bin equal to its mean value. For the next iteration, PCA is applied to the modified data matrix.

Step 7 : Repeat steps 1-6 until the maximum number of iterations is reached. The maximum is the only parameter that needs to be set in IPCA. The value of this parameter can change based on the number of the anomalies that are expected to be found in the time length of the input data trace.

In our time interval of interest (namely a two-hour data matrix), there can be barely 10 anomalies. The algorithm's stoppage point is thus defined to be 10 (i.e. after 10 iterations, the algorithm is terminated). We can then manually investigate and identify if there are any anomalies in these 10 suspicious time bins. Since the normal subspace in each iteration is constituted by only one principal component, large anomalies will not have the opportunity to take part in the normal subspace. As the algorithm progresses, the data matrix will be more monotonous in the sense that most of the data moves towards normal behavior as we remove the effect of the suspicious time bins.

Chapter 4

Traffic anomalies in network traffic flows

In this chapter, we introduce the enterprise's network traffic trace that we use for our simulations and the nature of the network traffic anomalies that we model. Moreover, we discuss the artificial injection these anomalies into the traffic trace in order to test the performance of our anomaly detector. IPCA is applied to traffic features collected from network traffic flows and not directly on raw packet trace data.

4.1 Trace-flow definition

A traffic flow is defined as packets that bear similar attributes. Different definitions of flow exist, each with differing granularity in terms of the types of traffic that constitute a single unique flow.

Table 4.1: The information that we obtain for each flow. Attributes in red color constitute the flow defining characteristics

Source IP address	Destination IP address	Source Port
Destination Port	Number of packets	Number of bytes
Start Time	End Time	Transport Protocol

In this thesis, a flow is defined as a stream or sequence of packets that has the *same* values for the following five attributes: source IP address, destination IP address, source port, destination port and the transport protocol (as shown in red color in Table 4.1). The rest of the information that we keep for each flow such as the number of packets and bytes, start time and end time is shown in Table 4.1 in black.

We use real traffic traces that represent traffic from a typical enterprise networking environment, as supplied by Solana Networks Inc. In Table 4.2, part of the traffic trace (consisting of two representative flows) is shown. The first attribute of each flow is marked in red color. The first flow of the table has source IP address 127.187.6.251,

Table 4.2: Example of flows (as shown in Table 4.1) in a traffic trace

127.187.6.251	127.180.90.248	8080
43681	4	1707
2012-11-24 00:04:30	2012-11-24 00:04:30	6
127.180.72.196	139.246.81.41	64741
80	1	48
2012-11-24 02:04:32	2012-11-24 02:04:32	6

destination IP address 127.180.90.248, source port 8080 and destination port 43681. It consists of four packets and 1707 bytes. It started on the 24 of November 2012 at 00:04:30 and ended on the 24th of November 2012 at 00:04:30. Its duration was hence less than a second. It is a TCP flow, so its transport protocol number is 6.

4.2 Modeling of network traffic anomalies

The aforementioned network traffic trace was infected with anomalies, using the FLAME tool [4], described in [31]. We used 10 different anomalies, outlined below. They are divided into three categories: Network Scans, Spams and DoS attacks. Based on the transport protocol, the anomalies can be further divided into TCP anomalies and UDP anomalies, as shown in Table 4.3.

Table 4.3: Modeled Anomalies

Protocol	Network Scans	Spams	DoS Attacks
TCP	Radmin Scan DCE-RPC SSH Scan	-	TCP Flood A TCP Flood B
UDP	Netbios Scan	Spam A Spam B	UDP Flood A UDP Flood B

The modeling of the anomalies is performed based on 10 flow attributes, following the approach of [4]. Those attributes are : transport protocol, source IP address, destination IP address, source port, destination port, flow size in packets, flow size in bytes, flow duration (in ms), flow inter-arrival times and TCP flags. However, TCP flags are not considered as attributes to our anomaly detection algorithm. Each of these characteristics is derived using different options from the FLAME tool. The transport protocol and the source IP address are fixed (constant) for each of the anomalies. The inter-arrival times of the anomaly flows follow the model described in [4]. We provide details about this modeling in the subsections below. As far as the other attributes are concerned, the three main anomaly groups share common functions with different parameters. Below, is a brief overview of the modeled traffic anomalies. These anomalies were modeled in [4], based on extracted flows caused by several malicious events from three weeks of Netflow traces captured in August 2007 at the SWITCH[1] border routers. For each event, a detailed characterization of all relevant flow attributes is presented. Of note, the duration of a flow is the time difference between the first and the last packet of the flow. The lowest time scale that is used in FLAME is milliseconds. Hence, a flow duration of 0 ms means that the start and end time of this flow is in the same ms (i.e. flow duration lower than 1ms).

[1]SWITCH is a non-profit organization providing backbone for the Swiss universities networks and NIC

4.2.1 Network scans

This group consists of four anomalies, three TCP anomalies (Radmin scan, DCE-RPC and SSH scan) and one UDP anomaly (Netbios scan). Based on the modeling in [4], the destination IP address starts from a random IP address. The next destination IP address is derived by adding an integer k (from the range [-k,k]) to the previous destination IP address. The value of k is 40 for Radmin Scan and 256 for DCE-RPC. For the Netbios scan, the increment is +1, but every 100 to 200 flows a decrement of 60 to 70 IP addresses occurs.

The destination IP addresses for SSH scan show a very irregular pattern. Periods of 30 to 200 flows scan a range of 400 IP addresses and at the end of each period a increment or decrement of 200 or 400 IP addresses occurs. The source port numbers are selected randomly from a set of possible port numbers (from 32000 to 61000) for the SSH scan. In the cases of Radmin scan and DCE-RPC, an integer (from -40 to 40 for Radmin Scan and from -200 to 200 for DCE-RPC) is selected as the increment or decrement of the next port number. Moreover, the port numbers are limited to [1K:5K] for the former and [1200:4800] for the latter. The source port number for the Netbios scan is fixed to a number above 10000.

As each network scan examines a specific port, the destination port number is fixed (22 for SSH, 4899 for Radmin, 135 for DCE-RPC and 137 for Netbios). The DCE-RPC and Netbios scan use a fixed number of packets and bytes, whereas the other anomalies in this group use a random number of packets and a constant number of bytes for each flow size. Flow durations (in ms) are selected randomly from a set of integers except when a flow contains one packet. In this case, the duration is 0 ms.

4.2.2 Spams

This category contains two variants of anomalies and they both use traffic using the UDP transport protocol [4]. Spam variant A selects a random destination IP address to start with and then chooses the next IP addresses randomly from a set of integers that ranges from -200 to 200 (as in the case of network scans). The same function is used for generating the source port number. The first source port number is selected from the range [32K:61K] and a increment of decrement in the range [-1K:1K] occurs in consecutive flows. After a constant number of flows (550), an increment of 2000 source port number occurs. Spam variant B selects a destination IP address from a block of 3000 IP addresses and after 300 flows the next block of IP addresses is used. As far as the source port numbers are concerned, the anomaly starts with a random source port number; after 550 flows, it increases the source port number by one to four ports. The destination port number for both of the anomalies in this group is either 1026 or 1027. Flow size is constant (1 packet, 925 bytes) which results in a constant duration of 0 ms (i.e. lower than 1 ms).

4.2.3 DoS attacks

The source and destination IP addresses for the DoS attacks are constant. The TCP DoS attacks also have a fixed destination port (80). The source port is selected randomly from within a range of ports (from 1K to 3K for variant A and from 49K to 65.4K for variant B). Flow size is constant for TCP variant A and flow durations is selected randomly between two values. Flow size for TCP variant B has two possible values (one packet with probability 0.264 and two packets with probability 0.736) and the duration for two packet flows is selected randomly for an interval. As far as the UDP DoS attacks are concerned, the source port numbers are selected randomly from within an interval (of length 19 for variant A and from 1 to 6K for variant B) and the destination port numbers are selected sequentially between 20 and 1024 for Variant A and randomly from the range 1k to 5K for Variant B. The flow sizes are constant for both anomalies to one packet resulting in 0 ms flow duration.

4.3 FLAME anomaly injection

The FLAME [31] framework is designed to artificially inject anomalies into traffic traces. It can also model a wide variety of traffic anomalies, including as additive (i.e. add extra flows in the trace), subtractive (i.e. subtract or modify existing flows from the trace) or a combination of the two. All of the anomalies that we experimented with fall into the first category (namely additive anomalies). Figure 4.1 shows the traffic anomaly injection process. FLAME takes a traffic trace, in Cisco Netflow v5 or v9 format [32] as an input, artificially injects anomalies into the trace and then exports the infected trace in Netflow v5 or v9 format. The modeling of the anomalies is performed inside FLAME via Python scripts. As a result, FLAME is a powerful tool for assisting with experiments for algorithms used to detect network traffic anomalies. By controlling the anomalies and the time that they are injected into the trace, one can know when to expect an anomaly when analyzing the infected traffic trace. This in turn allows one to effectively evaluate any algorithm used for traffic anomaly detection.

The main modules of FLAME that we used are *i)* the reader, which converts the Netflow trace into an internal format; *ii)* the flow generator, which is used for generating traffic flows (in our case, the additive anomalies); *iii)* the flow merger which is used to inject anomalies into the trace and *iv)* the writer, which performs the final step of converting the infected trace from the internal format to the Netflow format and extracting it. Figure 4.1 shows all the functional blocks and the process we follow in order to contaminate our network traffic trace. We first convert the database file that describes the network traffic trace to a format compatible with FLAME (Netflow v5) via a Python script. The reader takes the Netflow v5 trace file

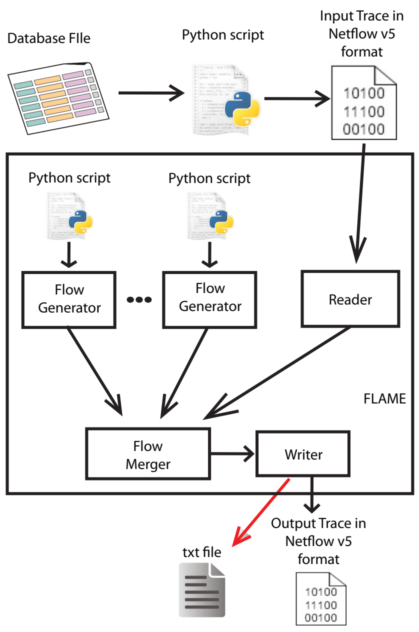

Figure 4.1: The traffic anomaly injection process

as input and converts it to an internal flow format. Python scripts are used to model the anomalies. We then inject the anomalies using the flow generator and flow merger modules. Each flow generator synthesizes flow streams that represent an anomaly. Anomalies are generated on flow level and not on packet level [4]. Through these flow generators, we can determine the number of flows for its anomaly. Hence, we can inject various network traffic anomaly intensities into our trace. The flow merger has two types of input. The first type is the stream of flows from each flow generator (i.e. each anomaly) and the second type is the reader that keeps the network traffic trace. The flow merger's output is the merged stream from both types of input. Merging is done based on flow packet timestamps. The writer module converts the infected trace from the internal FLAME format to Netflow v5. Finally, a text file (suitable as input to our IPCA algorithm) and a Netflow output are produced. We discuss further details about our experimental procedure in chapter 5.

Chapter 5

Experimental Procedure

In this chapter, we describe the process of generating artificially infected traffic traces, splitting a 24 hour network traffic trace into two-hour traces and grouping them by traffic load. The first step is the artificial injection of synthetic anomalies, which was described in Chapter 4, into a trace of real network traffic using the FLAME tool [31]. The second step is the conversion of the FLAME output into a text file (with the use of PYTHON scripts). This text file contains the 24 hour trace and the injected anomalies. We separate the raw network traffic trace (i.e. the trace without the artificial anomalies) from the anomalies with the use of MATLAB code via their pipeID (we describe this process in section 5.1). We then split the 24 hour raw network traffic trace into two-hour traces. Each two-hour trace is converted into a $Y_{m \times n}$ matrix which is the input to our algorithm, as described in Chapter 3. In our work, matrix Y captures the $m = 14$ traffic features, seven for TCP and seven for UDP traffic along $n = 96$ time bins (i.e. each time bin has duration of 75 seconds).

5.1 Framework description

The network trace that we acquired contains the traffic flows for 24 hours that we split in 12 two-hour traffic traces. Each traffic flow is defined by nine attributes (see Table 4.1). We consider flows of both TCP and UDP traffic.

As discussed in Chapter 4, the traffic trace is first converted (with the use of a PYTHON script) to a format compatible with FLAME (Netflow V5) and then inserted as an input into the FLAME tool. The modeling of the traffic anomalies and their artificial injection into a real traffic trace is implemented by means of the FLAME tool. The output file of FLAME is its input trace that is infected with the artificially created anomalies. The FLAME tool provides the ability to determine the number of anomalous flows to be injected for each modeled anomaly. We are hence able to infect different time bins with varying volumes of anomaly flows.

Table 5.1: The information that we obtain for every flow after the anomaly injection process

Source IP address	Destination IP address	Source Port	Destination Port	Bytes
Packets	Start time	End time	Transport Layer Protocol ID	Pipe ID

Table 5.1 shows the information we obtain for each network traffic flow after converting the output of the FLAME tool into a text file. We see that table contains the same information as Table 4.1 plus another attribute, the *pipeID* (shown in red color). The FLAME tool uses pipes in order to artificially inject anomalies into the trace. Each anomaly has a different $pipeID$. The first pipe contains the raw network traffic trace ($pipeID = 0$) and the rest of the pipes contain the artificial anomalies that are injected ($pipeID = 1$ for the first anomaly, $pipeID = 2$ for the second and so forth). The flows of all of the pipes are combined into a single file which is the output of FLAME. However, we can separate the anomalies from the actual trace via

their *pipeID*.

We can inject anomalies into any time bin of our choice by changing the start and end times of each anomalous flow for each anomaly. This offers us the advantage of utilizing the same contaminated trace and anomalies in different test scenarios (i.e. different contaminated time bins) which greatly facilitates and enables easy implementation. After the anomalies are injected, we split the 24 hour trace into 12 two-hour traces and apply our modified approach to each of them. Each two-hour interval is divided into 96 different time bins which means that the duration of a time bin is 75 seconds long.

5.2 Constructing two-hour traffic traces and grouping them by traffic load

Most of the traffic is during peak hours and traffic lessens as we approach the beginning or the end of the day. We thus expect that the number of traffic flows will be high in the traces that contain peak hour traffic and lower in the rest of the traces.

In order to infect the traces with varying anomaly flow percentages (the percentage of anomalous flows in a time bin), the 12 traces were further divided into 6 groups, according to the mean number of flows per time bin. This categorization is depicted in Table 4.1. Group #1 consists of traces {1,2,3} (00:00 - 06:00); Group #2 contains trace {12} (06:00 - 08:00, 18:00 - 22:00); Group #3 traces {4,10,11} (10:00 - 14:00); Group #4 traces {9} (08:00 - 10:00, 14:00 - 16:00); Group #5 contains trace {5,8} (16:00 - 18:00) and Group #6 contains traces {6,7} (22:00 - 24:00). Traces that are in the same group have the same average number of packets per time bin. For example, traces 4, 10 and 11 have on average 850 TCP flows per time bin. Recall that the time bin is 75 seconds long. In terms of network traffic intensity, Group

#3 traces (i.e. the three two-hour interval traffic that corresponds to traces 4,10 and 11 traffic is hence "created" at a rate of 11,33 flows/second. In each experiment, we

Table 5.2: Mean number of flows and packets per time bin

Protocol	Group#1	Group#2	Group#3	Group#4	Group#5	Group#6
TCP flows	250	400	850	1500	3000	4500
UDP flows	270	270	270	400	400	400
TCP packets	4000	$1.8 * 10^4$	$1.2 * 10^4$	$6 * 10^4$	$8 * 10^4$	$1.2 * 10^5$
UDP packets	876	975	962	1040	1070	1090

infected a trace with three anomalies each in a different time bin. The same three anomalies were used to infect one two-hour trace from each group which result in a total of six infected traces per experiment. We experimented with various intensities for the anomalies, from 5% to 90%. For the purposes of this thesis, the intensity of an anomaly is defined as the ratio of anomalous flows over the average number of flows per time bin. For example, in Group #2, an anomaly intensity of 10% is translated into 85 TCP or 27 UDP anomalous flows.

After the injection procedure, we built a MATLAB program to insert the anomalies into specific time bins. Each infected two-hour trace was then converted into a 96×14 matrix, the input to our detector, by aggregating the flow attributes of each time bin and calculating the entropies as described earlier.

Chapter 6

Experimental Results

Before providing details about the detection rates of the anomalies that we artificially injected in our traffic traces, we discuss the behavior of our algorithm when applied to the data traces before contamination. Since we do not have any control on the supplied traffic traces, we first run the algorithm on the "clean" traffic trace (i.e. before artificially injecting any anomalies). The time bins that will be identified as anomalous in the raw traffic trace will not be infected with any artificial anomalies. In this way, we eliminate any potential bias that may exist in the raw traffic data.

Before injecting artificial traffic anomalies, IPCA is applied to the traces and we mark the 10 time bins that the algorithm selected as being the most anomalous. We then contaminate with anomalies, time bins that do not belong to the aforementioned 10 time bins. In such a way, we are certain that if the algorithm detects the infected time bin, it is due only to the artificial anomaly.

We run IPCA for every two-hour trace that we obtained. The 10 time bins that the algorithm detected as suspicious in these 10 iterations are marked in red, in both the TCP and UDP figure. We also present the order in which these suspicious time bins were selected by the algorithm. Out of the 10 numbers that we provide under the figures of its experiment, the first number is the time bin that our algorithm detects as the most suspicious in the first iteration, the second is the time bin that it detects as the most suspicious in the second iteration and so forth. For example, assume that the time bins that our algorithm selected as potentially anomalous for a given traffic trace are: 15, 4, 8, 78, 20 , 11, 15, 1, 4 and 8. This means that time bin 15 was detected in the first iteration, time bin 4 in the second and so forth. Time bin 8 is the time bin that our algorithm detected in the 10th iteration. We also discuss the results and provide reasoning for the algorithm's behavior.

6.1 Good vs. bad traces

We separate our two-hour traces in "good" and "bad" traces. We provide details and reasoning about this separation by giving examples for each category and discussing their results, in subsections 6.1.1 and 6.1.2. "Bad" traces include a few of the two-hour traces of the enterprise networks that spanned non-office hours. Traffic in this traces is very scarce and the experiments were uninformative concerning the anomaly detection rates. On the other hand, "good" traces which mostly spanned office hours when traffic shows fewer irregularities were a good base for experimentation and validation purposes. When injecting traffic anomalies into these traces, the results were more robust and informative concerning the algorithm's detection capabilities. We provide examples for each category and discuss the reasons we feel this separation is essential below.

6.1.1 “Bad” traces

6.1.1.1 Traffic trace during 22:00h-24:00h

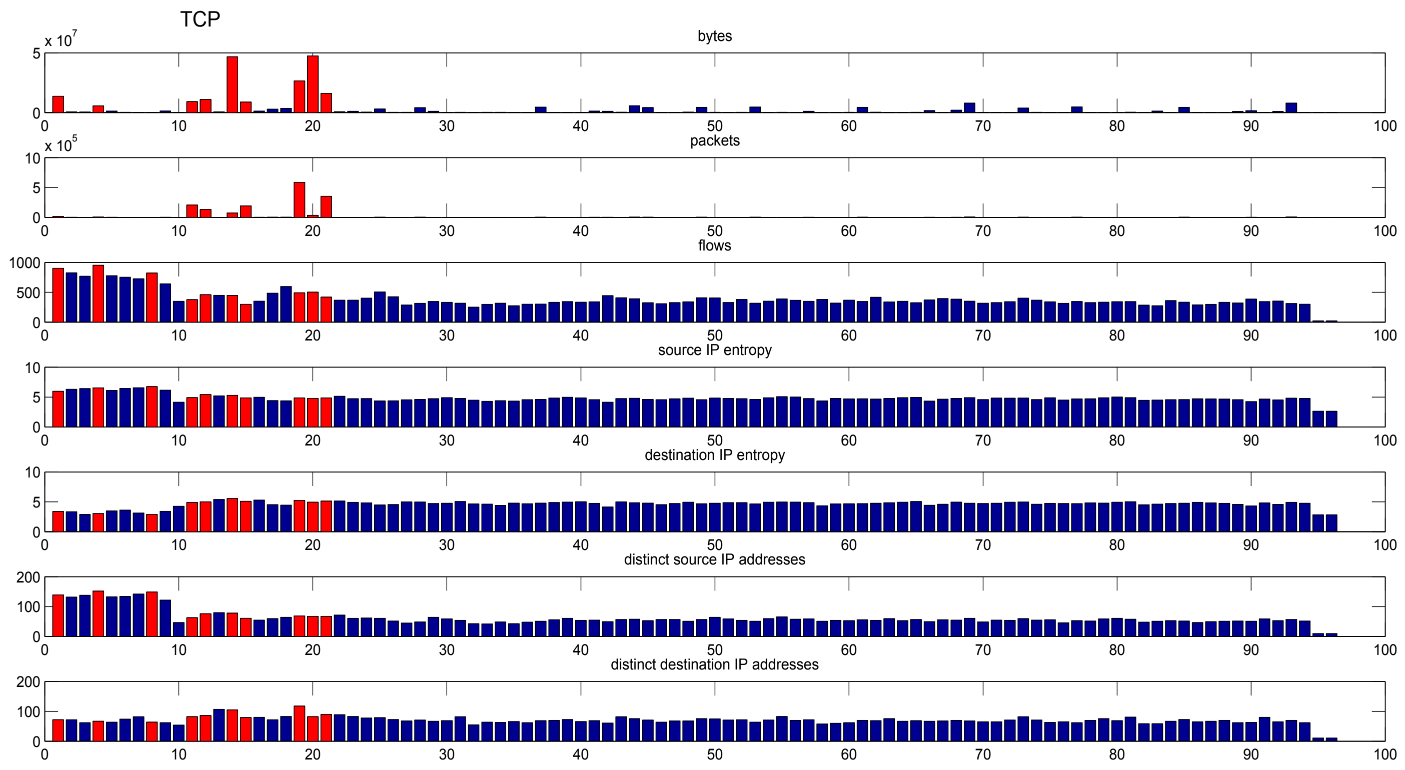

Figure 6.1: TCP traffic (22:00 - 24:00)

The time bins that our algorithm selected as potentially anomalous for this traffic traffic trace are: 12, 19, 21, 14, 20 , 11, 15, 1, 4 and 8. Out of all of the traces that we experimented with, this trace is probably the worst. As shown in Figure 6.1, TCP traffic is in general very sparse. This is not a surprise since there is very little traffic in an enterprise network during these hours. In most of the time bins, there are very few flows present. However, there are great spikes in some time bins, for example in 14, 20 and 19. On the other hand, UDP traffic, as shown in Figure 6.2, does not have such great spikes except in time bin 12. It is worth mentioning that our algorithm detects time bin 12 as the most anomalous in the first iteration, due to the UDP traffic. The rest of the time bins that the algorithm selects are due to the

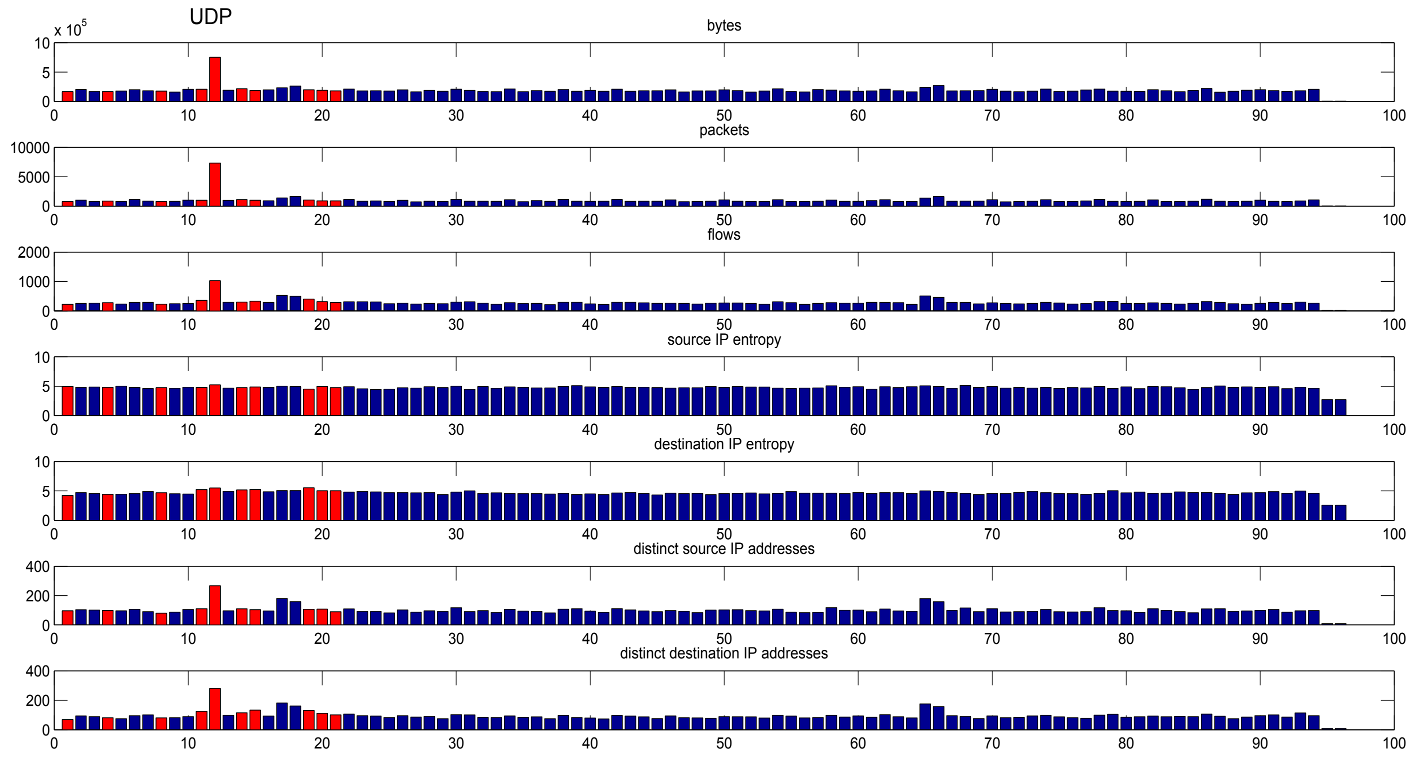

Figure 6.2: UDP traffic (22:00 - 24:00)

TCP traffic. We later infect this trace with anomalies by choosing to contaminate any time bin except the 10 aforementioned time bins that our algorithm detected as anomalous in the “clean” trace.

It subsequently becomes clear why we consider this a bad trace. For example, if we choose to contaminate time bin 40 with a TCP anomaly, the intensity of the anomaly would need to be significantly great, in order for the algorithm to detect it. This is the case for most of the time bins in this trace. If the anomaly is not large enough to cause spikes similar to at least time bin 8 (time bin selected in the 10th iteration), the algorithm will always choose the same 10 time bins that it chose before contamination, as the most anomalous.

6.1.1.2 Traffic trace during 04:00h - 06:00h

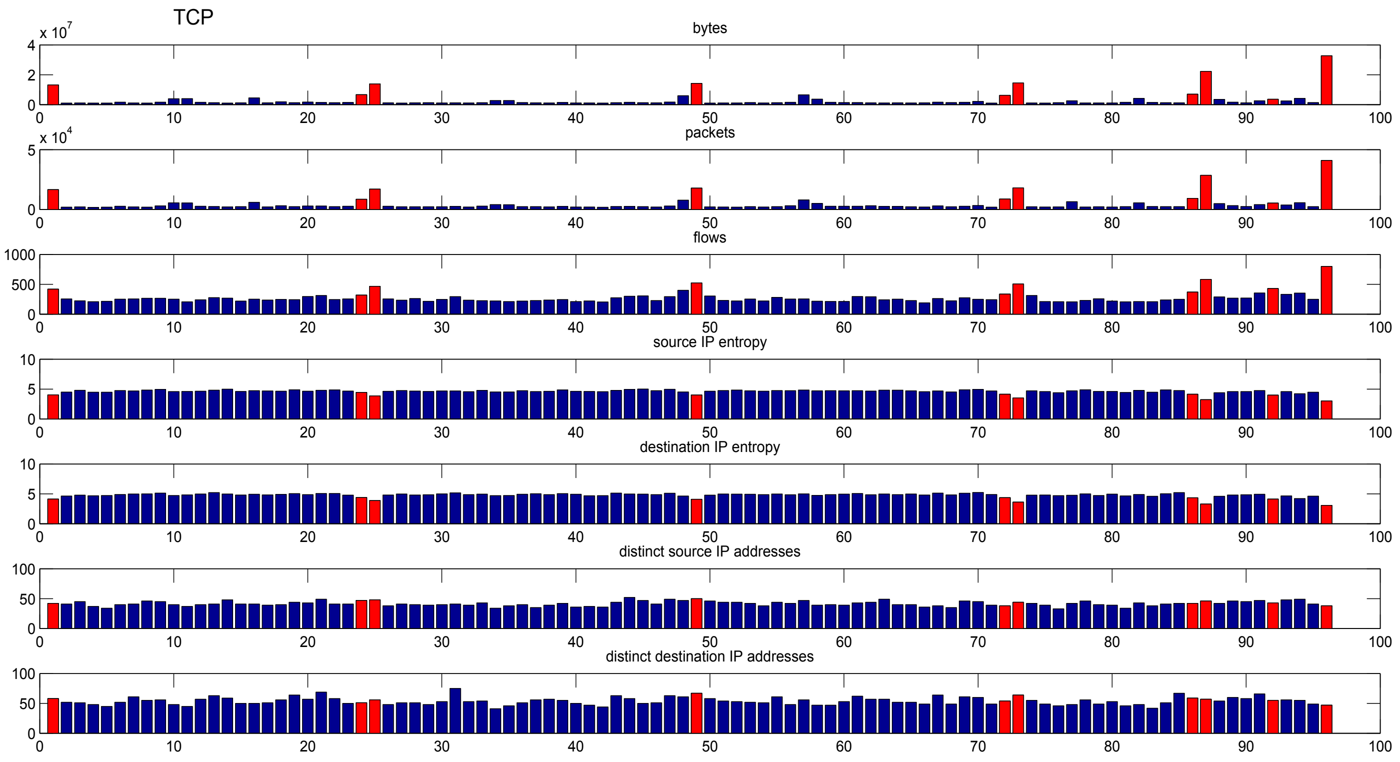

Figure 6.3: TCP traffic (04:00 - 06:00)

Figures 6.3 and 6.4 shows the TCP and UDP traffic for the two-hour interval 04:00-06:00. The time bins that our algorithm selected as potentially anomalous are: 96, 87, 73, 49, 25, 1, 86, 72, 92 and 24. Similarly to the previous example, TCP traffic is quite sparse. This is again expected as there is very little traffic in an enterprise network during these hours. The huge spikes in TCP traffic leads to every time bin that our algorithm selects as anomalous being due to the TCP traffic.

Although time bins such as 17, 18, 65 and 66 have significant spikes in UDP traffic, the algorithm does not select any of them. This is due to the fact that the TCP spikes are enormous when compared to the rest of the traffic. They are hence considered more anomalous than the large spikes in UDP traffic. If we were to contaminate the traffic with the UDP or TCP anomaly, the intensity would need to be heavy enough to cause spikes greater than these in time bin 24 (selected at the 10th iteration).

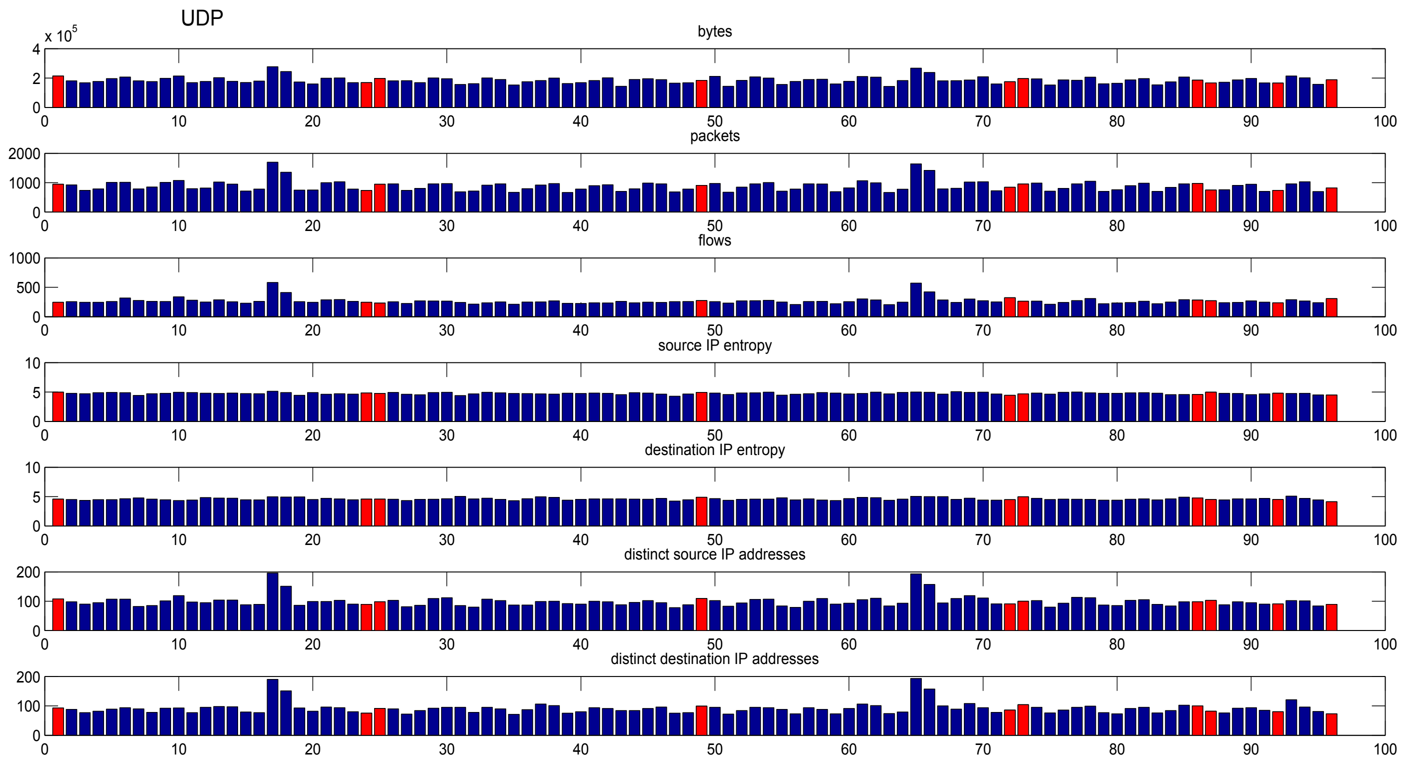

Figure 6.4: UDP traffic (04:00 - 06:00)

6.1.2 "Good" traces

6.1.2.1 Traffic trace during 12:00h - 14:00h traffic

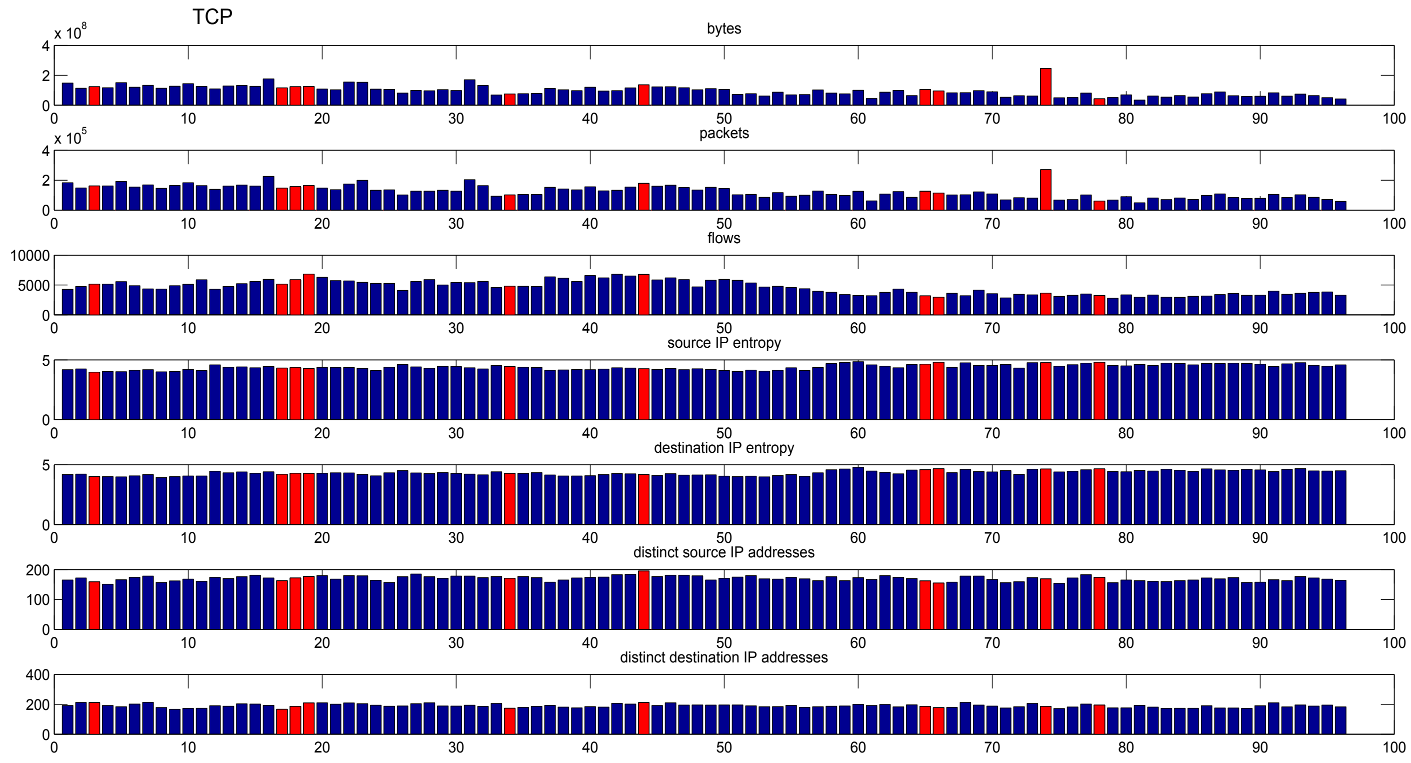

Figure 6.5: TCP traffic (12:00 - 14:00)

Figures 6.5 and 6.6 show the TCP and UDP traffic for the two-hour interval (12:00 - 14:00). The time bins that our algorithm selected as potentially anomalous are: 65, 17, 66, 18, 74, 34, 44, 19, 3 and 78. The difference between this traffic trace and the previous is clear; there are spikes both in both TCP and UDP traffic but not of the same magnitude as before. In such traces, we can effectively test the capabilities of our algorithms by injecting artificial anomalies. In this trace, most of the time bins in the 10 iterations are selected due to excessive UDP traffic. Only two time bins are selected due to TCP traffic: 74 is selected for excessive traffic in bytes and packets, while 44 is selected due to several source IP addresses being used.

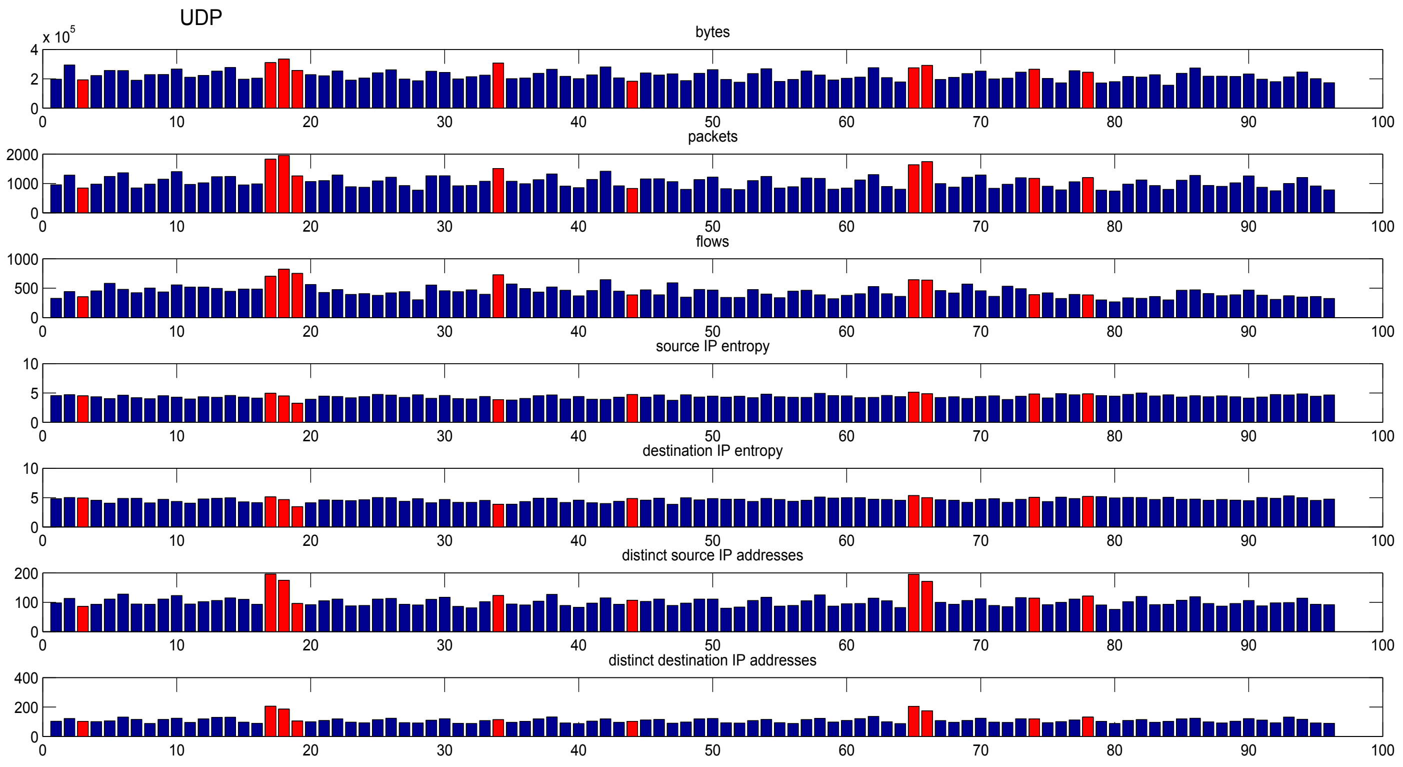

Figure 6.6: UDP traffic (12:00 - 14:00)

6.1.2.2 Traffic trace during 14:00h - 16:00h traffic

Figures 6.7 and 6.8 show the TCP and UDP traffic for the two-hour trace of the enterprise traffic between (14:00 - 16:00). The time bins that our algorithm selected as potentially anomalous are: 7 70, 69, 88, 95, 94, 61, 59, 3 and 14. The difference between this traffic trace and the previous "bad" traces is again clear. Once more, TCP traffic dominates the anomalous time bins. Apart from time bin 7, which is selected in the first iteration due to the huge spike in bytes, the rest of the time bins

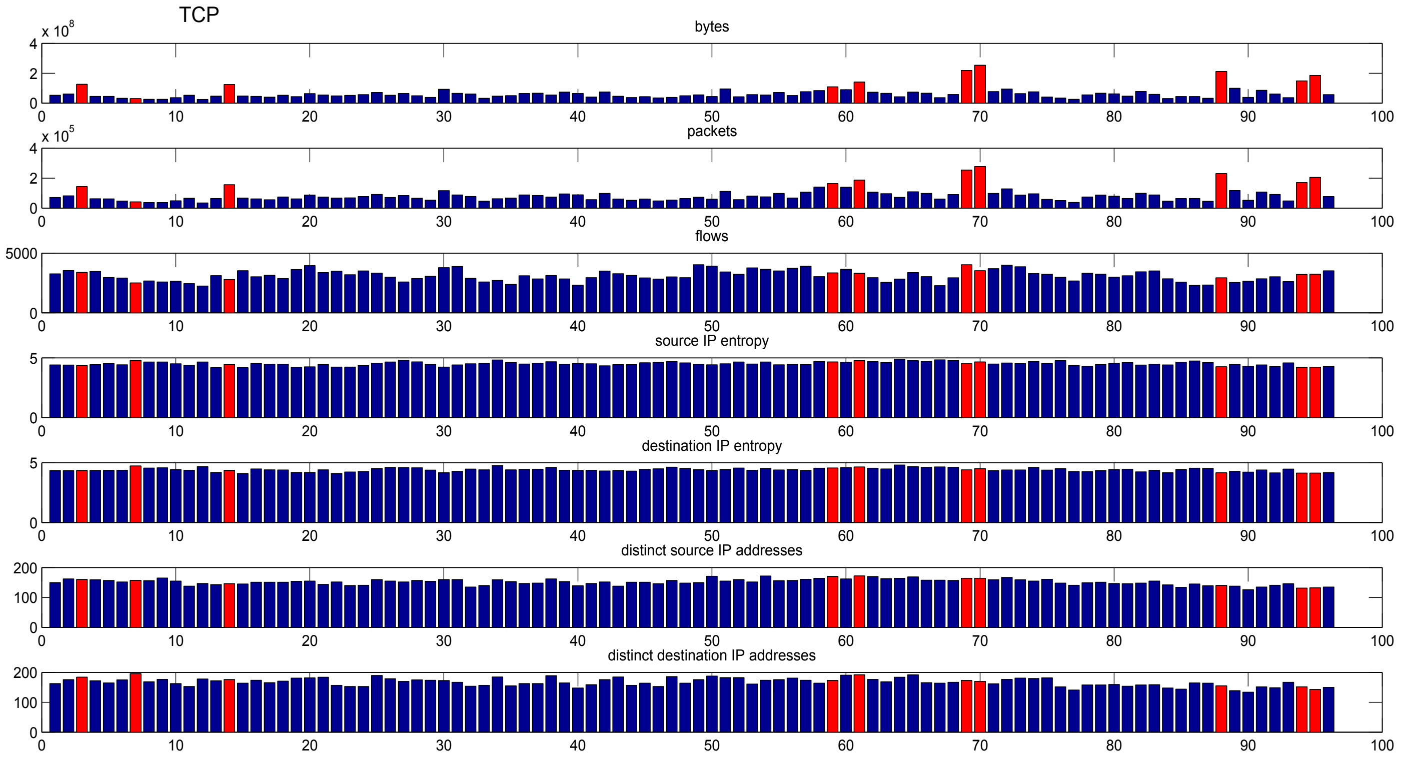

Figure 6.7: TCP traffic (14:00 - 16:00)

are selected due to the TCP traffic. Moreover, it is interesting that time bins such as 17,18, 65 and 66 are not selected despite the spikes in the UDP traffic. We provide reasoning for the behavior of our algorithm in the remainder of this chapter. We emphasize once more that we do not contaminate any time bin that was selected as anomalous when the algorithm was applied in the “clean” traffic trace.

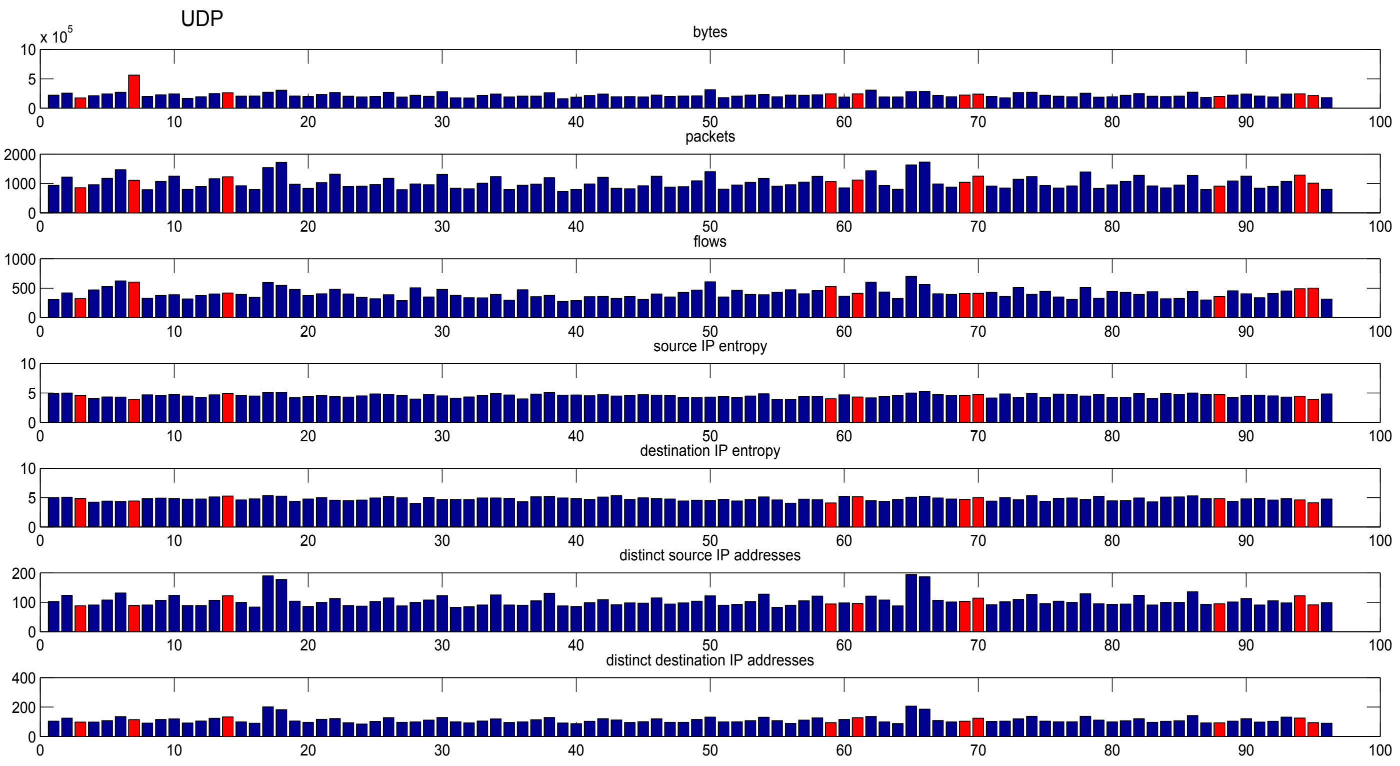

Figure 6.8: UDP traffic(14:00 - 16:00)

6.2 Experiments

Several different experiments were conducted. For every trace, different combinations of anomalies were injected into different time bins of the same trace. The results and general conclusion are presented in last section of this chapter. In this section, we focus on the most important experiments and provide insights into the artificial injection and the detection rates of the anomalies. We contaminate every trace with three anomalies, in different combinations for each experiment. For the same anomaly, we contaminate every time bin with different flow percentages, ranging from 5% to 90%. We experiment by injecting different anomalies into consecutive bins as well as into "distant" time bins, that is, bins that have at least 30 minutes of difference between each other. We find that this is irrelevant to the algorithm performance; in other words, anomalies do not influence each other, when injected into "neighbor" bins. In addition, we experimented by injecting different anomalies into the same time bin. However, we do not discuss about these experiments. The reason is that in these cases, the huge number of flows that were added into the same time bin made the detection easier than before. These experiments hence do not have any significant importance, in terms of IPCA efficiency.

6.2.1 Experiment 1: Contaminating the traces with three TCP Scans

In this experiment, we infected our trace with three TCP scans: Radmin scan, DCOM/RPC and SSH-scan. We contaminated six two-hour intervals. Table 6.1

Table 6.1: Experimental results for every traffic group

Anomaly percentage	Group#1	Group#2	Group#3	Group#4	Group#5	Group#6
5%	x	x	1	*all*	*all*	*all*
20%	$2, 3$	*all*	*all*	*all*	*all*	*all*
35%	*all*	*all*	*all*	*all*	*all*	*all*
50%	*all*	*all*	*all*	*all*	*all*	*all*
75%	*all*	*all*	*all*	*all*	*all*	*all*
90%	*all*	*all*	*all*	*all*	*all*	*all*

shows the results that we obtained. Within the table, "All" indicates that the algorithm detected all three anomalies, "x" indicates that it did not detect any and "1", "1,3" or "2" indicates that it detected those specific anomalies. For example, "1,3"

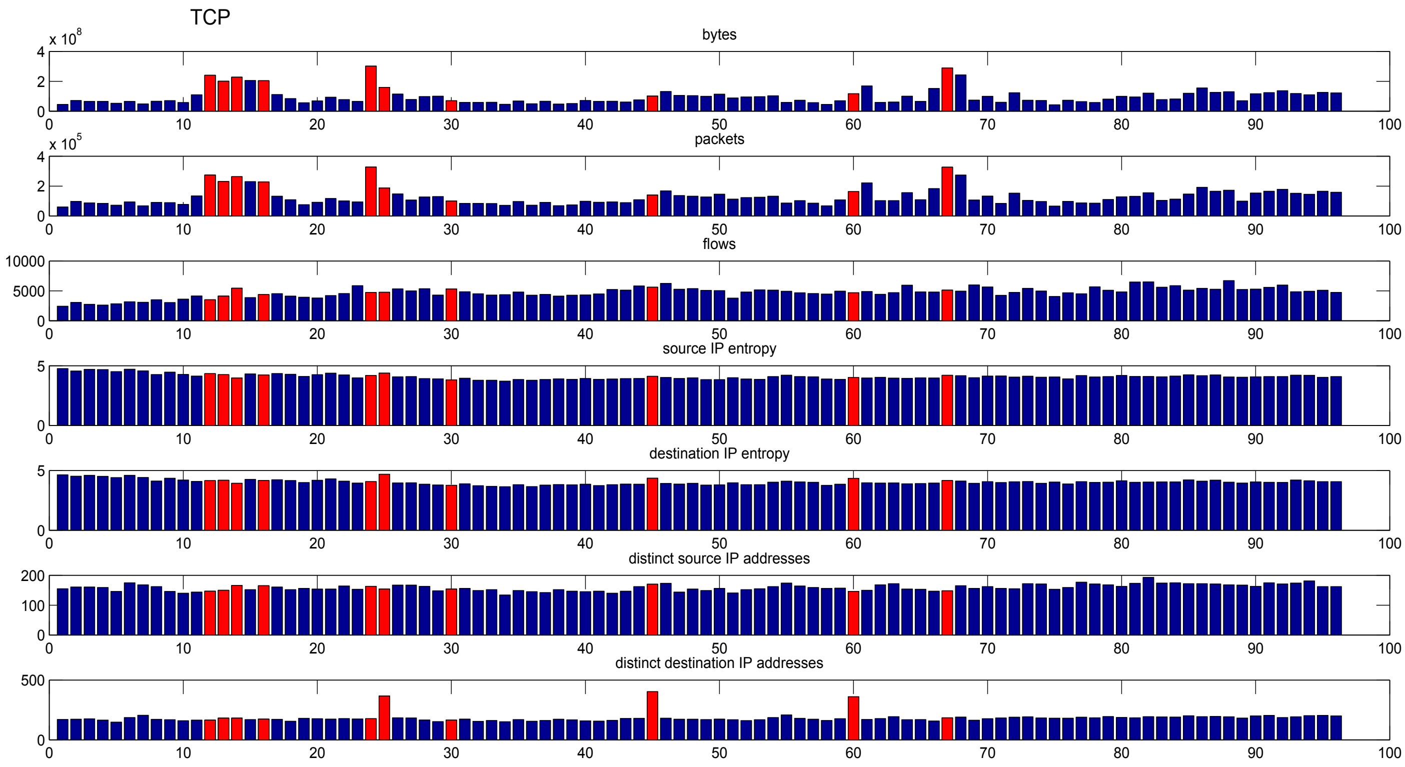

Figure 6.9: TCP traffic of trace {6} for 5% injected anomaly

means that the algorithm detected the first and the third anomaly (i.e the Radmin Scan and SSH-scan) but not the second (DCOM/RPC) while "2" indicates that only the second anomaly was detected.

The results of this experiment are very satisfactory. Even at very low percentages (5%) of anomaly injection , IPCA detects most of the anomalies. In general, scans were shown to be the easiest anomaly to detect. For an anomaly of 20% or higher, ICPA detects all three anomalies for almost every group. We next focus on some paradigms of these experiments and provide both figures and insight about the results. In Figures 6.9 and 6.10, the TCP and UDP traffic of trace {6} (Group #3), for an injected anomaly of 5% percentage is shown. We contaminated time bins 25, 45 and 60.

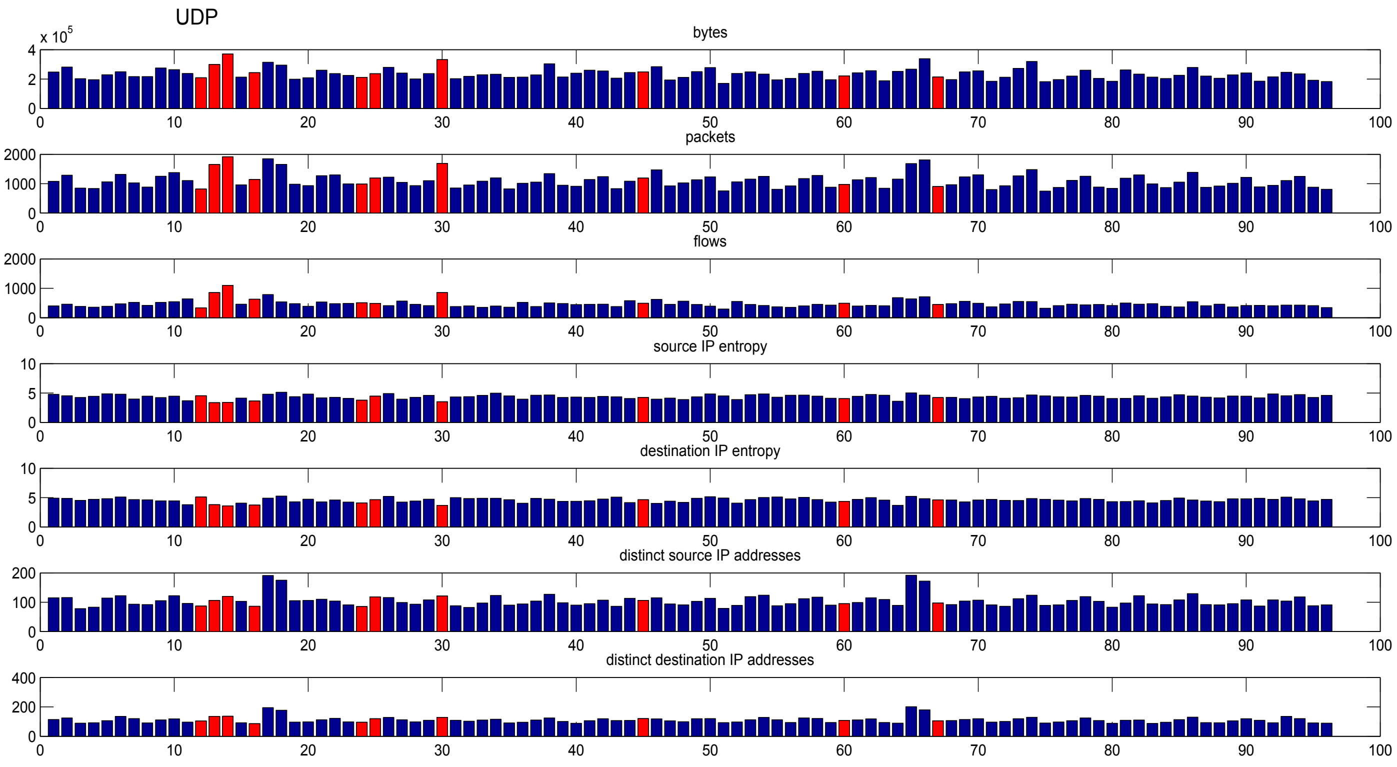

Figure 6.10: UDP traffic of trace {6} for 5% injected anomaly

The suspicious bin that algorithm detected were: 45, 25, 60, 14, 13, 30, 24, 67, 12, 16. We can see that IPCA detects these 3 anomalies in the first three iterations. For every anomaly, the seventh feature that corresponds to the distinct destination IP address is shown to have large spikes. This is the main reason that the algorithm detects all three anomalies despite the fact that we injected anomalies at a very low intensity. A distinct IP address is hence the most important feature when dealing with TCP scans (we later see that this is the case for UDP scans as well). In general, scans are the anomalies that are detected the most easily due to the huge spikes that they cause to the distinct destination IP addresses.

6.2.2 Experiment 2: Contaminating the traces with two TCP floods and one UDP Spam

In this experiment we infected our trace with two TCP floods(type-A and type-B) and UDP spam(type-A). The overall results are shown in Table 6.2:

Table 6.2: Experimental results for every traffic group

Anomaly percentage	Group#1	Group#2	Group#3	Group#4	Group#5	Group#6
5%	x	x	1	x	3	x
20%	x	x	$1, 3$	3	3	3
35%	$1, 3$	3	$1, 3$	$1, 3$	3	*all*
50%	*all*	3	*all*	*all*	$1, 3$	*all*
75%	*all*	*all*	*all*	*all*	*all*	*all*
90%	*all*	*all*	*all*	*all*	*all*	*all*

Obviously, our second anomaly (TCP flood type-B) is the hardest to be detected. The IPCA does not detect any flood anomalies in group #2, even when they are injected at a relatively great intensity. However, this is due to the inconsistency of the traces in this group, as described in section 6.1. In fact, floods are the hardest type of anomaly to detect as they do not cause significant volume changes in any feature. The algorithm's poor performance in floods forces us to undertake new experiment that includes the source and destination IP ports as features, as we see in section 6.3.

We again focus on some paradigms of these experiments and provide both figures and insights related to the results. In Figures 6.11 and 6.12, the TCP and UDP traffic of trace {4} (Group #3), for an injected anomaly of 35% percentage are shown. We contaminated time bins 11,56 and 89.

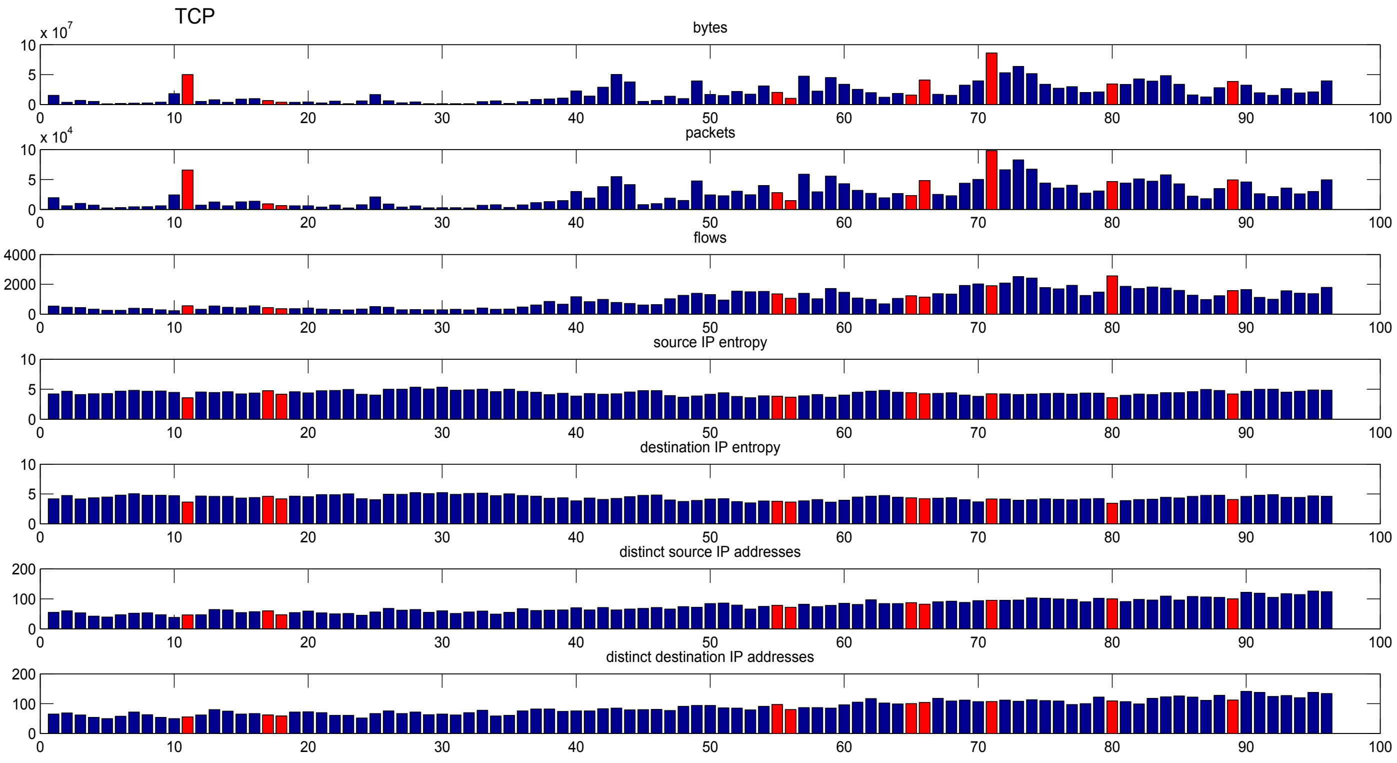

Figure 6.11: TCP traffic of trace {4} for 35% injected anomaly

The suspicious bins that the algorithm detected were: 17, 65, 55, 18, 66, 56, 89, 71, 11 and 80. Time bin 11 where we injected TCP flood type-A has enormous spikes in packets and flows. In addition, its source and destination IP entropy are shown to have small values in comparison to the neighboring time bins. This is due to nature of the anomaly that we injected. The source and destination IP addresses of flood type-A are constant (i.e. the same for every flow of anomaly).

This is hence the reason that both source and destination entropies in this time bin are small. In time bin 89, UDP anomaly is primarily detected due to the spikes in packets, flows and especially destination IP addresses. We chose to illustrate this

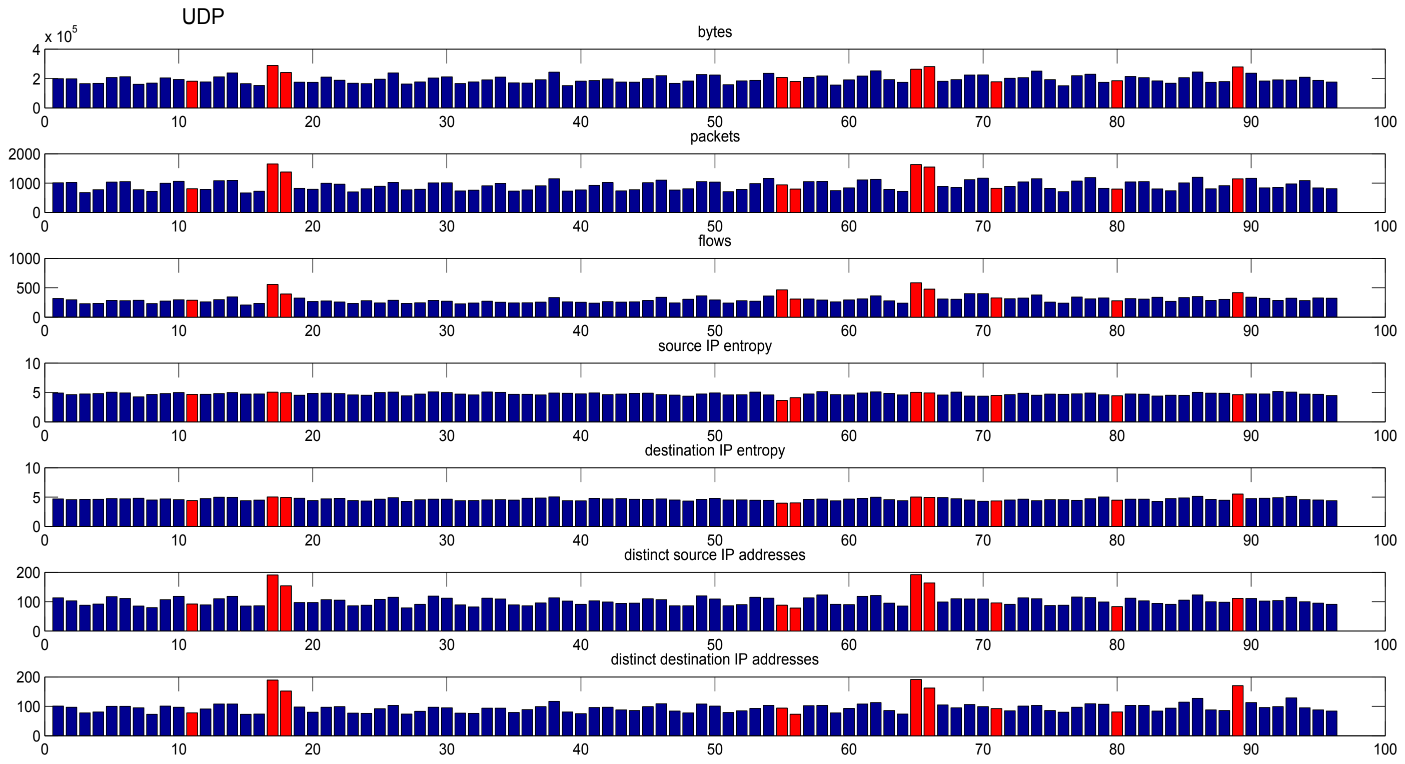

Figure 6.12: UDP traffic of trace {4} for 35% injected anomaly

paradigm for another reason. Time bin 56, where TCP flood type-B is injected, is selected in the sixth iteration. However, we do not consider this IPCA's success in detecting the anomaly due to the fact that this time bin is selected mostly due to the excessive UDP traffic. In other words, we show that time bin 56 is not a "good" time bin for testing IPCA's detection rates in TCP traffic since this time bin is dominated by the UDP traffic. This is also the reason that we undertook several experiments and tested the algorithm in different cases, infecting different time bins in every experiment.

6.2.3 Experiment 3: Contaminating the traces with two UDP Floods and one UDP Scan

In this experiment, we contaminate our traces with two UDP floods (UDP flood type-A, UDP flood-B) and one UDP scan(Netbios). The UDP scan is anomaly "2" in Table 6.3.

Table 6.3: Experimental results for every traffic group

Anomaly percentage	Group#1	Group#2	Group#3	Group#4	Group#5	Group#6
5%	x	x	2	x	x	x
20%	x	x	$1, 2$	2	2	2
35%	2	$1, 2$	$1, 2$	$1, 2$	$1, 2$	2
50%	*all*	$1, 2$	*all*	*all*	*all*	*all*
75%	*all*	*all*	*all*	*all*	*all*	*all*
90%	*all*	*all*	*all*	*all*	*all*	*all*

The UDP scan is detected more easily than the two UDP floods ("2"). Even at a very low intensity percentage (5%), it is detected in Group #3. The hardest anomaly to detect is UDP flood type-A ("3") where in some cases, we need to inject 75% or more in order for it to be detected. After TCP floods, UDP floods are the second most difficult type of anomaly to detect . We again focus on some paradigms of these experiments and provide both figures and insights related to the results.

In Figures 6.13 and 6.14, the TCP and UDP traffic of trace {7} (Group #6), for injected anomaly of 50% percentage is shown. We contaminated time bins 11,56 and 89.

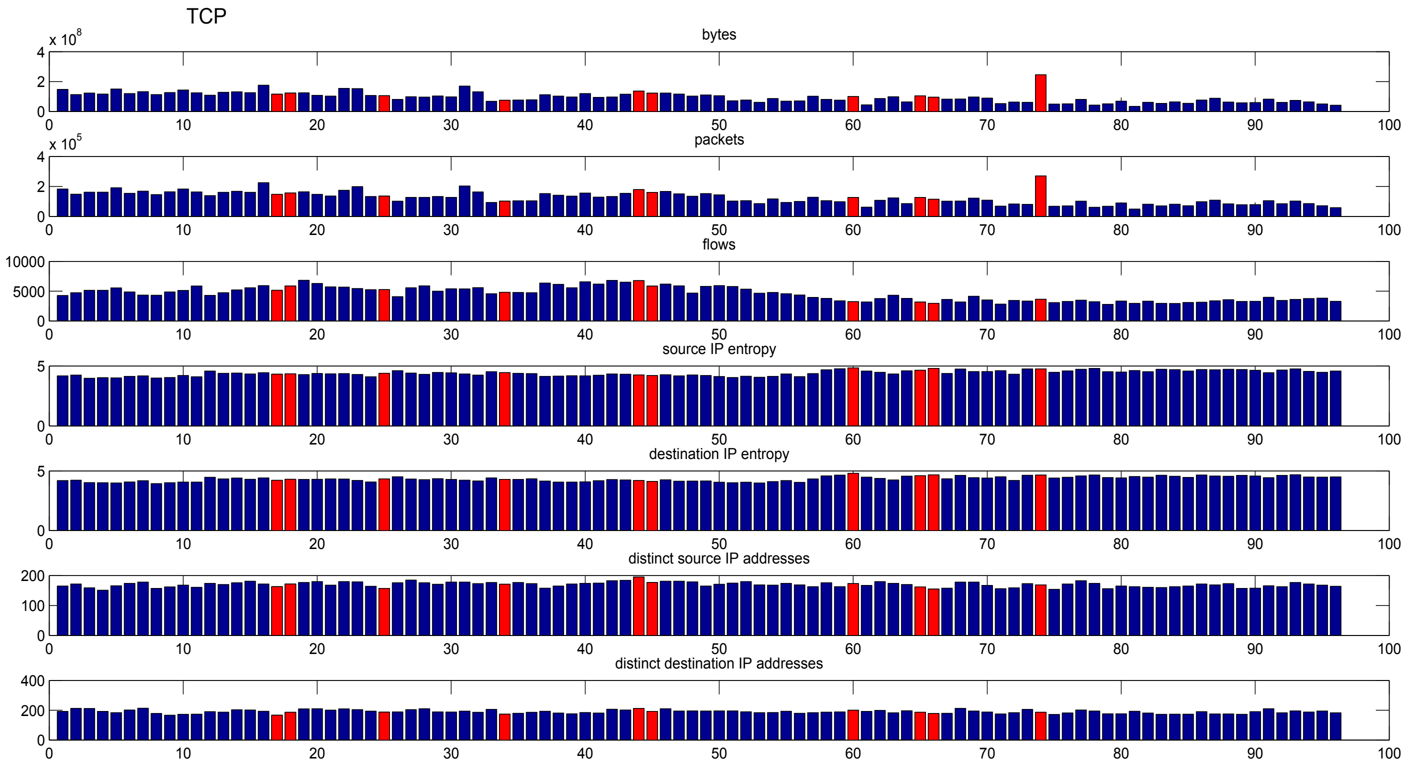

Figure 6.13: TCP traffic of trace {7} for 50% injected anomaly

The suspicious bins that the algorithm detected were: 45, 65, 17, 66, 18, 74, 25, 34, 44 and 60. We infected time bins 25,45 and 60. In the first iteration, UDP scan is detected due to the huge spike in the distinct destination IP address feature. However, this is not the case for the UDP floods. Although the anomaly's traffic intensity is relatively great, UDP floods are detected in the 7th and 10th iteration. This is because they do not cause such large spikes as the UDP scan. However, the spikes (which are mostly in bytes, packets and flows) are sufficient for their detection.

It subsequently becomes clear why the detection rates are better when a injecting a UDP scan instead of a UDP flood anomaly.

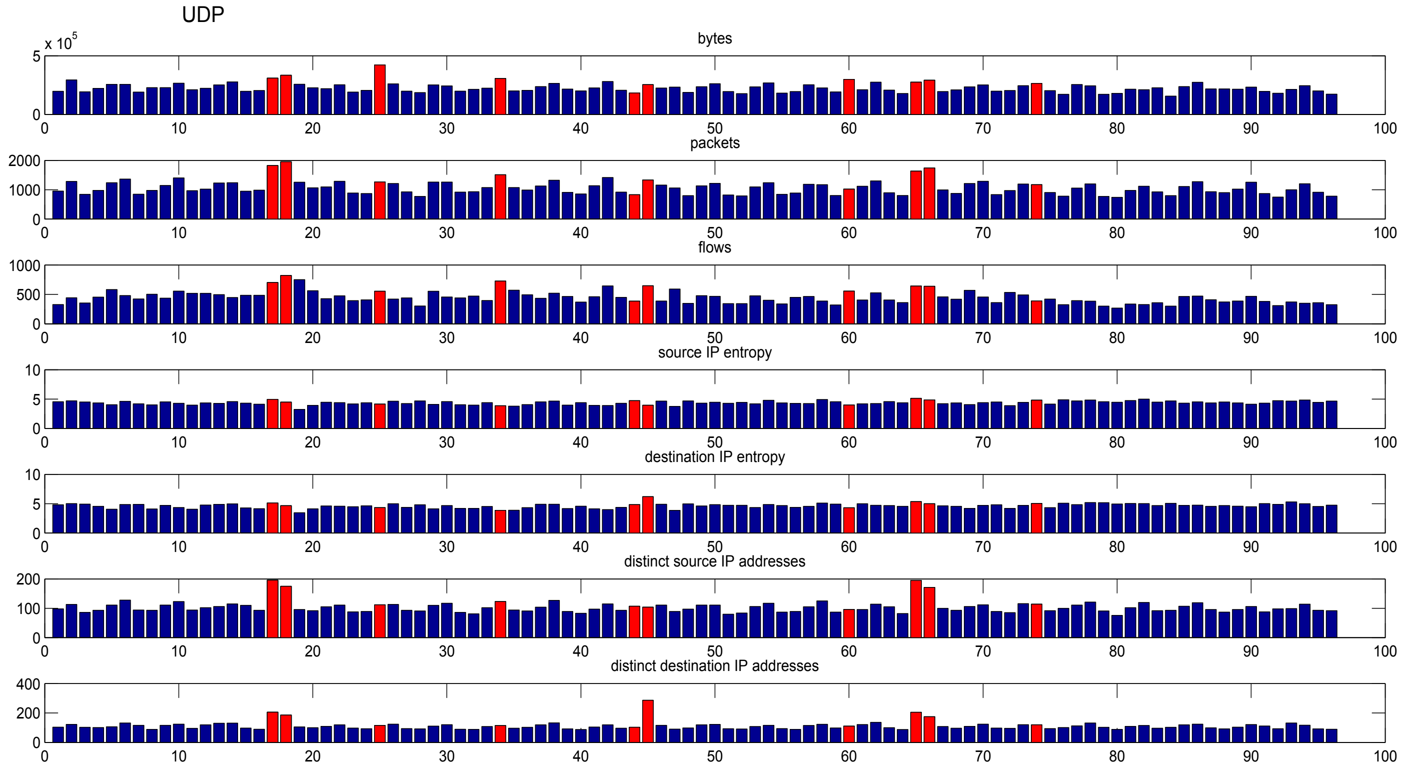

Figure 6.14: UDP traffic of trace {7} for 50% injected anomaly

6.3 Anomaly detection and validation

In this section, we define the minimum injected percentage of anomalous flows that resulted in successful detection. Figure 6.15 shows the lower and upper limits of the intensity threshold for which each anomaly category is detectable, with respect to the traffic load (i.e. for every traffic group). Each category contains a number of anomalies (four Scans, two Spams, four Floods). The upper limit suggests that if

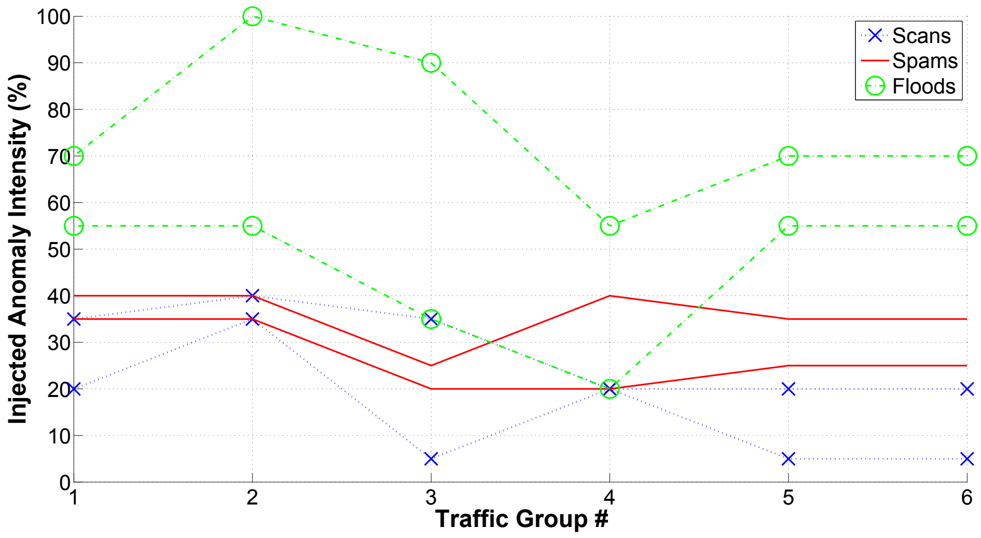

Figure 6.15: IPCAs anomaly detection limits

we inject an anomaly with an intensity greater than that, IPCA will surely detect it while anomalies with intensity below the lower detection limit are undetectable. An intensity of 100% means that the attack was *undetectable*. If an anomaly is detected at a certain intensity, then it will surely be detected at any higher intensity (up to 90%). As we can see from Figure 6.15, the detector performs better for Groups 3 to 6, where the traffic load is elevated. This can be justified by that fact that during off-peak hours, the traffic is scarce. Bursts of traffic create spikes in the features distribution that can overshadow the injected anomalies. (especially if anomalies are injected into time bins with minimal traffic). They are thus flagged as potential anomalies. During peak hours, the traffic feature distributions are much smoother and an anomaly is more easily detected. The algorithm works extremely well in the cases of network scans where it can sometimes detect from an intensity as low as 5% and in most cases it can detect anomalies from intensities of 20% and more. Similar results, with a slightly higher intensity as the lower boundary, are found for the network spams. As far as the DoS attacks (both for TCP and UDP) are concerned, we can see that the algorithm does not perform relatively well. The lower boundary for detection is highly increased in comparison to scans and spams. Furthermore, DoS attacks seem to be “stealthier” and are sometimes undetectable!

Our initial results motivated us to identify suitable features that will efficiently characterize certain anomalies. We thus introduce some extra features while eliminating others, depending on the nature of the anomaly that we want to detect. Since the number of bytes, packets and flows will bring an increase in the total traffic for every anomaly type and therefore indicate a potential anomaly, we decided to leave these features untouched. Each anomaly flow of network scans and spams is targeted to a different IP address which results in an increased number of distinct destination IP addresses. The dominant feature for detecting such anomalies is thus the distinct destination IP addresses. Each anomalous flow of DoS attacks is generated from a different port. Moreover, in the case of UDP DoS attacks, each anomalous flow is in different destination port. Therefore, attributes of the ports might be helpful in detecting such anomalies. Furthermore, each anomalous flow of network scans and spams is generated from a different port. By including port features, the detection of all anomalies could thus be enhanced.

Based on the above observations, the necessity of including source and destination port features in IPCA is obvious. We employed three modifications in the feature selection in order to enhance the anomaly detection rates of IPCA. In all cases, the total number of features was kept constant. In the first modification (mod 1), we replaced the source and destination IP entropy with those of the source and destination port entropy. In the second (mod 2), we replaced the attributes of the destination IP address with that of the source port. As we discussed earlier, the destination IP address features are vital for detecting network scans and spams. Finally, in the third modification (mod 3), we included port instead of IP address features. The last two modifications were implemented in order to enhance detection for DoS attacks only. We present the results in Figures 6.16, 6.17 and 6.18. These figures show the minimum lower threshold for which an attack was detectable (with and without the three modifications).

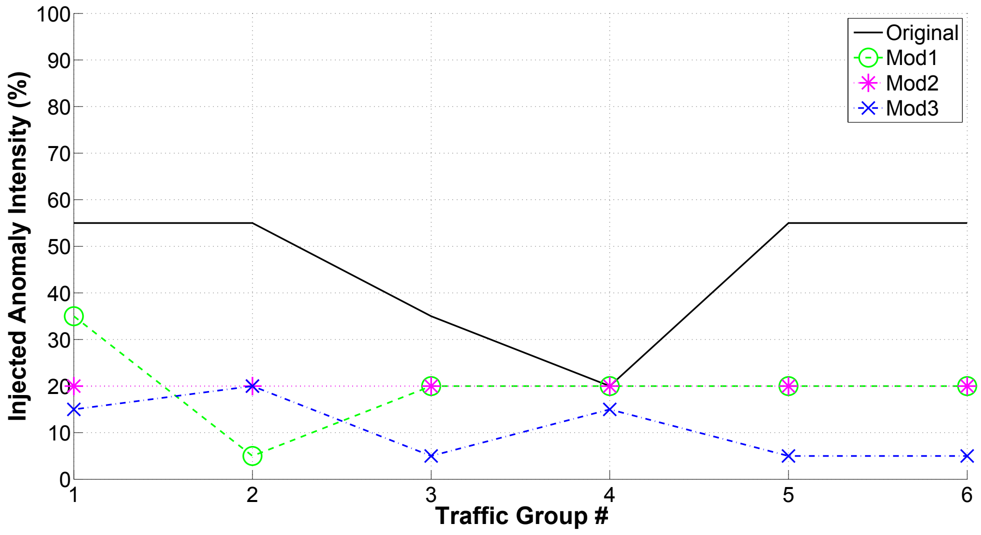

Figure 6.16: IPCAs Anomaly Detection Limits with modifications - DoS attacks

From Figure 6.16, we can see that these modifications greatly improve the detection capabilities in DoS attacks. In all of the modifications, DoS attacks are always detected if they are injected at 20% intensity or higher, except for in modification 1, Group 1. Traffic features concerning ports are hence proven to be useful in the detection of DoS attacks. Moreover, the first and second modification have almost the same effects on the detection process and differ only in the first two groups (where the traffic is scarce). As mentioned earlier, anomaly detection in these two groups highly depends on the time bins that are infected due to bursty traffic (i.e. traffic with large spikes). Moreover, the third modification (which includes only port features) can greatly enhance the detection capabilities of IPCA. When the third modification is applied, DoS attacks are detected even when the anomaly is only of 5% intensity! The results for Network Scans and Spams are shown in Figure 6.17 and 6.18, respectively.

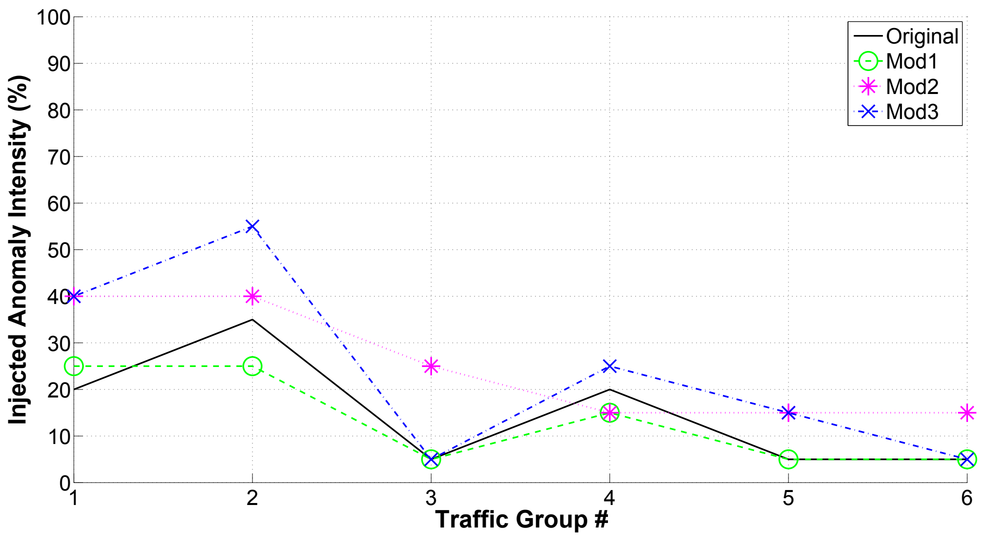

Figure 6.17: IPCAs anomaly detection limits with modifications - Network Scans

The first modification resulted in slightly lowering the thresholds for scans and slightly increasing them for spams. The other two modifications tend to elevate the thresholds for these two anomaly types. Although the third modification is shown to effectively confront DoS attacks, the first modification enhances the detection capabilities of DoS attacks while keeping the detection capabilities of the other attacks almost intact, which result in a better overall performance.

Figures 6.19 and 6.20 show the experimental probability of detection, for each anomaly category, based on the intensity of the injected anomaly,respectively, without and with feature modifications. The red lines represent the probability of detection for the DoS attacks, the dashed line the third modification and the thick line the original features. The green line shows the results of the original features for network

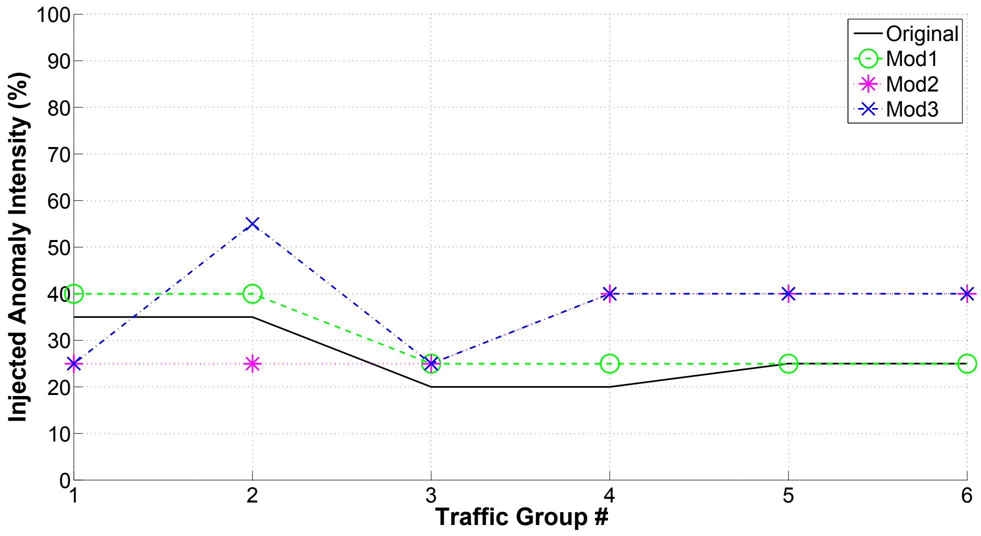

Figure 6.18: IPCAs anomaly detection limits with modifications - Network Spams

spams and the blue line those of the first modification for network scans. In particular, Figure 6.20 shows the results of the modification with the best detection probability (among all modifications), for eachy anomaly. The detection probability of network scans is increased when the first modification is applied (Figure 6.19). Without any modification (Figure 6.20), network scans are fully (100%) detected for intensity beyond 35%. With the first modification, network scans are fully detected beyond 25% intensity; they are even detected at low intensity (5%) with probability of 40%. As discussed earlier, 25% intensity contributes very little extra traffic to the total trace (only about 0.25%!). In the case of network spams, the original features outperform any modifications. We therefore do not include any further change in feature selection for this traffic anomaly. Network Spams are fully detected at intensities of at least 40% and have a high detection probability at intensities greater than 30%. The third modification greatly improves the detection probability of DoS attacks. We can see that the probability of anomaly detection is increased by 20% at the lowest

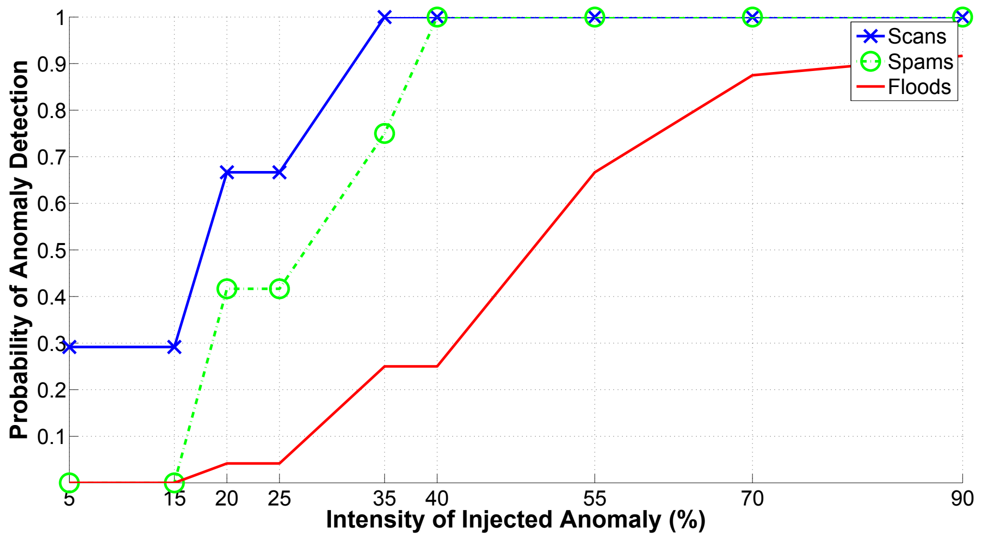

Figure 6.19: IPCAs probability of detection - no modifications

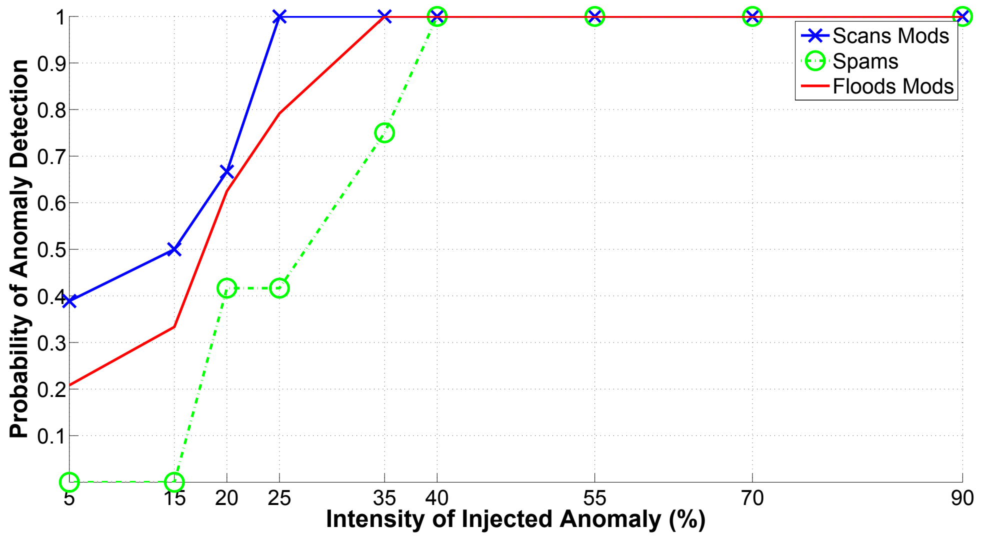

Figure 6.20: IPCAs probability of detection with modifications

intensity and by 75% at 35% intensity. Furthermore, when DoS attacks are injected at intensities of 35% or more, they are fully detected.

6.4 PCA vs IPCA probability of anomaly detection comparison

Three versions of PCA were implemented, using one, two and three eigenvectors as the normal subspace (PCA 1, PCA 2 and PCA 3, respectively). All algorithms use the traffic features of the original implementation of IPCA. Figure 6.21 compares the performance of IPCA and PCA. No SPE threshold was defined for the PCA algorithm.

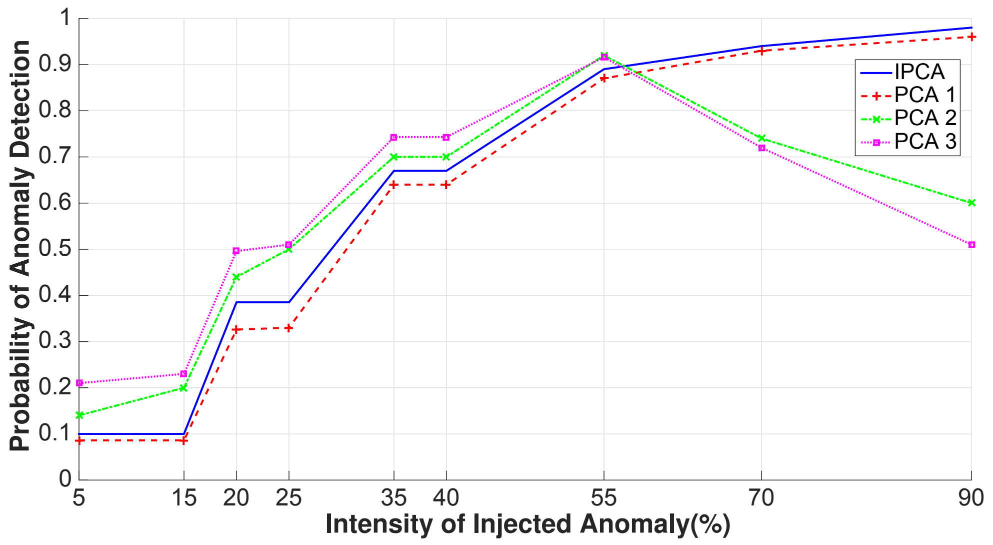

Figure 6.21: Comparison between IPCA and PCA1, PCA2, PCA3

Instead, the 10 most anomalous bins were flagged. This allows for a fair comparison between the algorithms, as they both flag the same number of time bins. As we can see from the figure, IPCA slightly outperforms PCA 1 due to redefining the normal subspace at each iteration. When comparing with PCA 2 and 3, we can see that for low intensities PCA has a higher probability of detection. But after 55% intensity, we can see that the performance drops. This is due to normal space poisoning, i.e. large anomalies are included in the first eigenvectors and are not detectable. IPCAs main advantage is that it provides us with a tool flexible to the input data and free from the many parameters calibration of PCA. In addition, the results are comparable and even better, at some cases, with those of PCA.

Chapter 7

Conclusions and Proposal

In this thesis, we have discussed the problem of network anomaly detection. As research shows, PCA is a widely known approach in this field yet the classical approach has proven to be sensitive to the calibration of its parameters and to normal subspace poisoning.

We introduced a modification of the PCA algorithm, namely the IPCA, in order to overcome these limitations. This method can be used as an initial tool for the detection of anomalies based on the statistical analysis of data traces. We model and inject artificially created anomalies into a trace and evaluate our approach. The performance of IPCA has shown promise for effective anomaly detection. Regarding the appropriateness of the selected features, some anomalies can be efficiently detected using no modifications while others, such as floods, require modification in the selected traffic features. In fact, experiments show that IPCA can detect most of the anomalies, even in the case of low flow anomaly intensities. Nonetheless, we experienced a difficulty in detecting DoS attacks, due to the nature of the anomaly which is why we replaced some traffic features of the algorithm with features concerning the source and destination ports. Doing so enabled us to significantly improve the results. We can specifically derive the following conclusions for the network traffic anomalies:

TCP scans: They are detected by our algorithm at a very sufficient rate (even at 5% of network traffic anomaly intensity in the "good" traces) due to the fact that they cause a spike in the distinct destination IP addresses feature.

UDP scan (Netbios): It is detected by our algorithm at a very sufficient rate (from an intensity as low as 5%) mostly due to the fact that it causes a spike in the distinct destination IP addresses feature, similar to the TCP scans.

UDP spams: In most of the cases, they are detected by our algorithm when the anomaly intensity is of 20% or more. However, due to the spike in the distinct destination IP addresses feature, there can be detected at even lower rates (in some cases, they were detected from an intensity as low as 5%). For scans and spams, the distinct destination IP addresses feature is hence shown to be the most "important" in terms of network anomaly detection rates.

TCP floods: Floods in general show the worst results. They do not cause a significant change to any of our "original" features (i.e. before modifications) and have low detection rates. However, our modifications show that the distinct source port and the source IP port entropy can play a significant role towards detecting these anomalies. Especially in the third modification (i.e. distinct source and destination ports), the anomaly is detected at very low anomaly intensities of 5% whereas in our "original" traffic feature setup, the algorithm detects the anomaly only after we raise the anomaly flows to 55%. Our feature modifications show that floods can also be detected in small scales, provided that we have included the "appropriate" set of traffic features in our traces.

UDP floods: Similarly to the TCP floods, the fact that they do not cause a significant change to any of our "original" traffic feature setup makes it difficult for the algorithm to detect them. However, our modifications show that the distinct destination port and the source and destination IP port entropy can play a significant role towards detecting these anomalies. When these features are included, UDP floods can be detected in low percentages of traffic volume intensity (20% or even 5%).

In regards to IPCA, our modified PCA approach, we can derive the following conclusions. Its strongest feature is its iterative approach. In every iteration, IPCA finds the most anomalous bin and replaces it with an "average" bin in terms of network traffic intensity (i.e. average number of bytes, average number of flows and so forth for every traffic feature). Hence, at every iteration, the most anomalous bin is removed and our trace is "smoothed". That allows us to detect anomalies that are not significantly large in terms of volume (in the latter iterations). Since the most anomalous bins are removed in the earlier iterations, even a small irregularity in a time bin can trigger detection. In the latter iterations, IPCA is able to detect very small irregular patterns that would be impossible to detect if these largely anomalous bins were not replaced by average time bins in the early iterations.

The larger an irregularity in a time bin is, the sooner (i.e. in the earlier iterations) IPCA will choose this time bin as anomalous. If an anomaly is not significantly large in volume, it may be detected in the latter iterations (i.e. after our algorithm detects and replaces the extremely irregular time bins). We could hence select the number of iterations based on what type of anomalies we want to detect. If we target large scale anomalies, the first three to four iterations would suffice. However, if we target anomalies that do not cause a significant change in any traffic feature, a larger number of iterations is needed.

Contrary to the classical approach, we form the PCA normal subspace by selecting only the first eigenvector. In such a way, IPCA effectively attacks the problem of normal subspace poisoning. Even at extreme cases where the network anomaly intensity is very high (i.e. 90%), the anomalies do not contaminate the normal subspace.

They remain in the abnormal subspace and are hence detected.

For each traffic trace, we construct two matrices, one for TCP and one for UDP traffic. Our approach treats both the TCP and UDP traffic of a time bin as unified traffic. In some cases where only one of them (either TCP or UDP traffic) shows significant variance, this may be a problem. If for example our TCP trace is highly variant, then a UDP anomaly will be harder to be detected since it will be "burried" under the TCP traffic variance.

The features that we select in order to apply our algorithm must be carefully considered. In fact, bytes, packets and flows show sometimes patterns in a "redundant" way. An anomaly (or a large spike in general) in one of these three features sometimes results in similar spikes in the other two features. In future work, perhaps one or two of these features could be removed and replaced by other. We could experiment for example with the ratio of bytes over flows or the ratio of packets over flows for every time bin. Hence, an anomaly that causes large spikes in bytes but not if packets or flows could be potentially detected due to these ratios. In addition, experiments show that the source and destination IP addresses entropy have a very "smooth" pattern (i.e. very few spikes). In extreme cases where we injected a large volume anomaly in a time bin, there were spikes in both source and destination IP entropies. However, these spikes are in most cases proportional to a larger spike in bytes, packets and flows and distinct source and destination IP address. Hence, if a spike exists in entropies due to an anomaly, there are even larger spikes in one or more of the remaining features. That leads us to believe that perhaps the source and destination IP addresses entropies are not as "important" features as the distinct source and destination IP address in terms of network anomaly detection rates.

In future work, more anomalies anomaly types should be modeled and tested. Furthermore, we will experiment on the second phase of network anomaly detection, the identification phase. In fact, preliminary results show that applying the same method inside the anomalous time bins enables us to detect the IP addresses that are the root of these anomalies. Hence, we could potentially identify the type of network anomaly that causes the irregularities and take the appropriate countermeasures. In conclusion, a method that can be fully automated, can certainly include elements of the IPCA algorithm. Future work should look into this area and see how IPCA can be combined with other implementations of anomaly detection, such as wavelets, as the first layer or network anomaly defence.

List of References

[1] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Computing Surveys (CSUR)*, vol. 41, no. 3, 2009.

[2] S. Staniford, J. A. Hoagland, and J. M. McAlerney, "Practical automated detection of stealthy portscans," *Journal of Computer Security*, vol. 10, no. 1, pp. 105–136, 2002.

[3] C. De Stefano, C. Sansone, and M. Vento, "To reject or not to reject: that is the question-an answer in case of neural classifiers," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 30, no. 1, pp. 84–94, 2000.

[4] D. Brauckhoff, *Network Traffic Anomaly Detection and Evaluation.* PhD thesis, ETH, 2010.

[5] A. Lakhina, K. Papagiannaki, M. Crovella, C. Diot, E. D. Kolaczyk, and N. Taft, "Structural analysis of network traffic flows," *SIGMETRICS Perform. Eval. Rev.*, vol. 32, no. 1, pp. 61–72, 2004.

[6] A. Lakhina, M. Crovella, and C. Diot, "Mining anomalies using traffic feature distributions," in *ACM SIGCOMM Computer Communication Review*, vol. 35, pp. 217–228, 2005.

[7] C. Pascoal, M. Rosario de Oliveira, R. Valadas, P. Filzmoser, P. Salvador, and A. Pacheco, "Robust feature selection and robust pca for internet traffic anomaly detection," in *INFOCOM, 2012 Proceedings IEEE*, pp. 1755–1763, 2012.

[8] T. Hastie, R. Tibshirani, and J. Friedman, *Unsupervised learning.* Springer, 2009.

[9] O. Sheluhin and A. Pankrushin, "Measuring of reliability of network anomalies detection using methods of discrete wavelet analysis," in *Science and Information Conference (SAI), 2013*, pp. 393–397, Oct.

[10] H. Li and Q. Wu, "Research of clustering algorithm based on information entropy and frequency sensitive discrepancy metric in anomaly detection," in *Information Science and Cloud Computing Companion (ISCC-C), 2013 International Conference on*, pp. 799–805, Dec 2013.

[11] T. Ahmed, M. Coates, and A. Lakhina, "Multivariate online anomaly detection using kernel recursive least squares," in *26th IEEE International Conference on Computer Communications. IEEE*, pp. 625–633, May 2007.

[12] R. Kwitt and U. Hofmann, "Unsupervised anomaly detection in network traffic by means of robust pca," in *Computing in the Global Information Technology, 2007. International Multi-Conference on*, pp. 37–37, March 2007.

[13] C. Callegari, L. Gazzarrini, S. Giordano, M. Pagano, and T. Pepe, "A novel pca-based network anomaly detection," in *Communications (ICC), IEEE International Conference on*, pp. 1–5, June 2011.

[14] D. Brauckhoff, K. Salamatian, and M. May, "Applying pca for traffic anomaly detection: Problems and solutions," in *INFOCOM*, pp. 2866–2870, IEEE, 2009.

[15] R. Jiang, H. Fei, and J. Huan, "A family of joint sparse pca algorithms for anomaly localization in network data streams," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 25, pp. 2421–2433, Nov 2013.

[16] L. Huang, X. Nguyen, M. Garofalakis, J. Hellerstein, M. Jordan, A. Joseph, and N. Taft, "Communication-efficient online detection of network-wide anomalies," in *26th IEEE International Conference on Computer Communications*, pp. 134–142, May 2007.

[17] H. Ringberg, A. Soule, J. Rexford, and C. Diot, "Sensitivity of pca for traffic anomaly detection," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 35, pp. 109–120, 2007.

[18] Y. Liu, L. Zhang, and Y. Guan, "Sketch-based streaming pca algorithm for network-wide traffic anomaly detection," in *Distributed Computing Systems (ICDCS), IEEE 30th International Conference on*, pp. 807–816, June 2010.

[19] D. Liu, C.-H. Lung, N. Seddigh, and B. Nandy, "Entropy-based robust pca for communication network anomaly detection," in *Communications in China (ICCC), IEEE/CIC International Conference on*, pp. 171–175, Oct 2014.

[20] T. Kudo, T. Morita, T. Matsuda, and T. Takine, “Pca-based robust anomaly detection using periodic traffic behavior,” in *Communications Workshops (ICC), IEEE International Conference on*, pp. 1330–1334, June 2013.

[21] B. Zhang, J. Yang, J. Wu, D. Qin, and L. Gao, “Pca-subspace method x2014; is it good enough for network-wide anomaly detection,” in *Network Operations and Management Symposium (NOMS)*, pp. 359–367, April 2012.

[22] N. H. Duong and H. Dang Hai, “A semi-supervised model for network traffic anomaly detection,” in *Advanced Communication Technology (ICACT), 17th International Conference on*, pp. 70–75, July 2015.

[23] Y. Zhang, P. Calyam, S. Debroy, and M. Sridharan, “Pca-based network-wide correlated anomaly event detection and diagnosis,” in *Design of Reliable Communication Networks (DRCN), 11th International Conference on the*, pp. 149–156, March 2015.

[24] L. Liu, X. Jin, G. Min, and L. Xu, “Real-time diagnosis of network anomaly based on statistical traffic analysis,” in *Trust, Security and Privacy in Computing and Communications (TrustCom), IEEE 11th International Conference on*, pp. 264–270, June 2012.

[25] M. Ahmadi Livani and M. Abadi, “A pca-based distributed approach for intrusion detection in wireless sensor networks,” in *Computer Networks and Distributed Systems (CNDS), International Symposium on*, pp. 55–60, Feb 2011.

[26] C. Issariyapat and K. Fukuda, “Anomaly detection in ip networks with principal component analysis,” in *Communications and Information Technology. ISCIT. 9th International Symposium on*, pp. 1229–1234, Sept 2009.

[27] G. Androulidakis, V. Chatzigiannakis, and S. Papavassiliou, “Using selective sampling for the support of scalable and efficient network anomaly detection,” in *Globecom Workshops*, pp. 1–5, Nov 2007.

[28] H. Kim, S. Lee, X. Ma, and C. Wang, “Higher-order pca for anomaly detection in large-scale networks,” in *Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP), 3rd IEEE International Workshop on*, pp. 85–88, Dec 2009.

[29] S. Hakami, Z. Zaidit, B. Landfeldt, and T. Moors, “Detection and identification of anomalies in wireless mesh networks using principal component analysis

(pca)," in *Parallel Architectures, Algorithms, and Networks. I-SPAN. International Symposium on*, pp. 266–271, May 2008.

[30] S. Novakov, C.-H. Lung, I. Lambadaris, and N. Seddigh, "Combining statistical and spectral analysis techniques in network traffic anomaly detection," in *Next Generation Networks and Services (NGNS)*, pp. 94–101, Dec 2012.

[31] D. Brauckhoff, A. Wagner, and M. May, "Flame: A flow-level anomaly modeling engine.," in *CSET*, 2008.

[32] "Netflow services and applications." White Paper. `http://ehealth-spectrum.ca.com/download/netflow.pdf`.

[33] G. Wilkinson, *Identification of Hostile TCP Traffic using Support Vector Machines.* PhD thesis, Oxford University, 2009.