Ann. Telecommun. (2014) 69:363–377
DOI 10.1007/s12243-013-0412-5

On the testing of network cyber threat detection methods on spam example

Robert Filasiak · Maciej Grzenda · Marcin Luckner · Pawel Zawistowski

Received: 15 March 2013 / Accepted: 13 November 2013 / Published online: 9 January 2014

Abstract As a response to the increasing number of cyber threats, novel detection and prevention methods are constantly being developed. One of the main obstacles hindering the development and evaluation of such methods is the shortage of reference data sets. What is proposed in this work is a way of testing methods detecting network threats. It includes a procedure for creating realistic reference data sets describing network threats and the processing and use of these data sets in testing environments. The proposed approach is illustrated and validated on the basis of the problem of spam detection. Reference data sets for spam detection are developed, analysed and used to both generate the requested volume of simulated traffic and analyse it using machine learning algorithms. The tests take into account both the accuracy and performance of threat detection methods under real load and constrained computing resources.

Keywords Network Intrusion Detection Systems (NIDS) · Flow analysis · Spam detection · Network data sets

R. Filasiak · M. Grzenda
Orangel Labs Poland, Obrzeżna 7, 02-691 Warszawa, Poland

R. Filasiak
e-mail: Robert.Filasiak@orange.com

M. Grzenda
e-mail: M.Grzenda@mini.pw.edu.pl

M. Grzenda · M. Luckner (✉)
Faculty of Mathematics and Information Science,
Warsaw University of Technology, Koszykowa 75,
00-662 Warszawa, Poland
e-mail: mluckner@mini.pw.edu.pl

M. Grzenda
e-mail: Maciej.Grzenda@orange.com

P. Zawistowski
Faculty of Electronics and Information Technology,
Warsaw University of Technology, Nowowiejska 15/19,
00-665 Warszawa, Poland
e-mail: P.Zawistowski.2@elka.pw.edu.pl

1 Introduction

1.1 Insufficient number of network data sets and its impact on cyber security

Modern high bandwidth core networks are susceptible to numerous cyber threats. Among them, large-scale spam distribution and Distributed Denial of Service (DDoS) [11, 23] attacks are of significant importance. Any kind of network forensics aimed at the reduction of these threats includes their prior detection.

When trying to implement efficient countermeasures to mitigate the scale of threats, local system administrators, network operators and research groups are faced with a variety of possible intrusion detection approaches, methods and off-the-shelf systems. This is accompanied by a very limited number of up-to-date network data sets that could be used to verify the cyber security of network systems at different levels. This makes the validation of individual threat detection methods problematic, if it is possible at all. In turn, the security of the systems is negatively affected. Moreover, the research into efficient techniques of intrusion detection is largely suffering for the same reason, i.e. limited availability of realistic network data sets [7]. In their recent survey [19], Tavallaee et al. report that 144 out of 276 analysed anomaly-based intrusion detection methods proposed over the last few years used Knowledge Discovery in Databases

(KDD) and Defense Advanced Research Projects Agency (DARPA) data sets for evaluation purposes. The latter data sets, in spite of their past contribution to intrusion detection research have at the same time been extensively criticised [19]. The main reason why these data sets, which originated several years ago, are still used is the lack of other, publicly available data sets. The need for sharing data sets is still observed and widely discussed in the network community. In particular, it was reported to be one of the major concerns during the Fifth Workshop on Active Internet Measurements hosted in 2013 by the Cooperative Association for Internet Data Analysis (CAIDA). Among other reasons, the need for data sharing to parameterise or validate models and inferences and to establish historical baselines of network behaviour [5] was emphasised.

Taking into account these data challenges, it is not surprising to see that relatively well-known intrusion methods are still successfully being used to compromise the security of many systems.

At the same time, what is especially promising is the ability to detect cyber threats at a network level via Network Intrusion Detection Systems (NIDS). This could result in more efficient digital evidence processing performed by the network operators, i.e. on a wider scale than the scale of individual network hosts. However, it is even more difficult to validate different intrusion detection techniques in this case. In the case of NIDS, additional legal constraints have to be considered, which limits the number of feasible techniques. On the one hand, customers expect their Internet Service Providers (ISPs) to provide efficient, fully secured access to the Internet. On the other hand, requirements related to privacy and confidentiality are equally important.

Moreover, cyber threats are constantly evolving. Therefore, the validation of both existing and newly developed NIDS should be based on up-to-date representative network data sets. This suggests the need to develop methods and procedures making possible the generation of such data sets based on real network traffic but not affecting the privacy of users causing this traffic.

Among other purposes, such data sets could be used by different research centres to establish a common baseline for the evaluation of the usability of individual misuse detection techniques. The use of only internally available data sets obviously prohibits the development of ground truth. In the latter context, the representation of network traffic via NetFlow records containing discriminators such as those proposed in [16] can be promoted as it does not reveal any packet data. Moreover, network flows can be additionally anonymised by changing IP addresses.

To sum up, rather than just a few new network data sets, an entire process of data set generation is needed. A high level overview of such a process is presented in Fig. 1. First, the data from real network traffic is acquired and filtered, e.g. based on a protocol and/or ports. Individual events such as HTTP requests or mail transmission are labelled to mark the events representing threats of interest. Next, the data can be anonymised. Finally, it can be converted to another format such as NetFlow. Such conversion can be applied at different stages of the process depending on the category of threat and the data needed to label it.

1.2 The need for the combination of data sets and testing environments

One more obstacle affecting the adoption of novel detection methods in high bandwidth networks is the limited attention paid in many novel methods to performance aspects. In many works, the tests of different intrusion detection techniques are made in an off-line manner using the data files as an input for individual techniques. Hence, the performance of individual techniques is not tested. At the same time, in high bandwidth networks, the performance of the traffic capturing process is a challenge which can be even more demanding when machine learning algorithms have to be executed in parallel in the same environment.

For all these reasons, a combination of realistic network data sets and testing environments using network emulation methods [15] is needed. This should be used to test both the accuracy of threat detection techniques and their performance. Ideally, the performance should not only be tested under different load conditions such as the overall volume of analysed network traffic but also the number of parallel

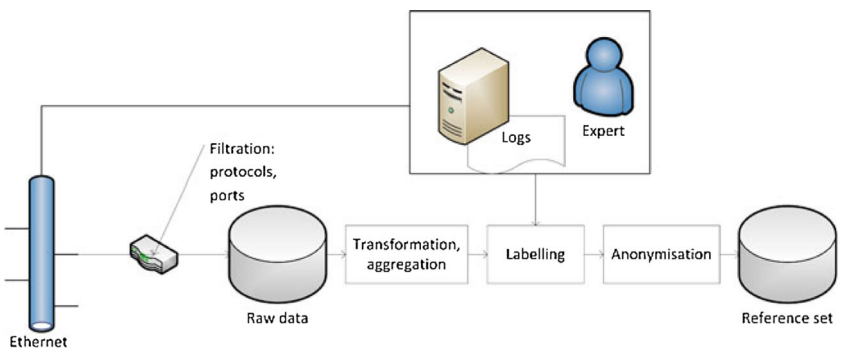

Fig. 1 The reference set creation

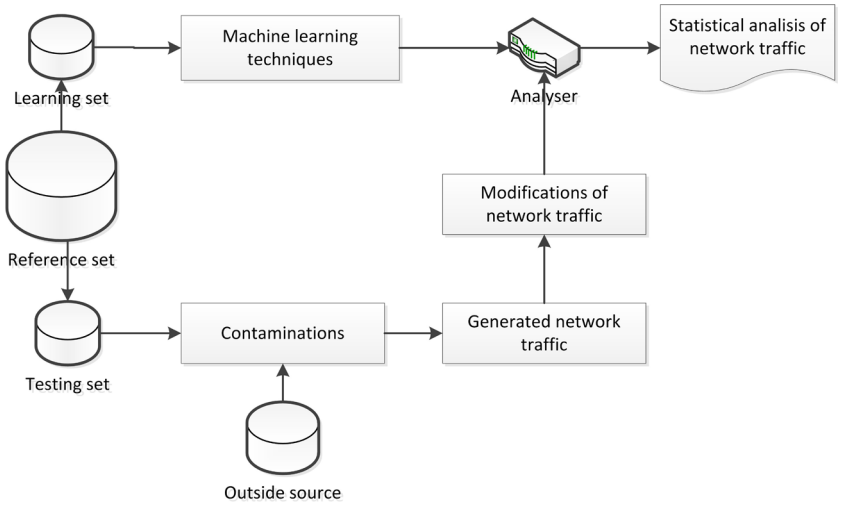

Fig. 2 The test bed

network flows or the average size of network datagrams. To sum up, a combination of all these factors, i.e. the methods of creating up-to-date network data sets when needed, the use of representative data sets based on real network traffic, the ability to scale the volume of this traffic and the ability to test an end-to-end solution including network traffic probe(s), data aggregation and threat detection based on the data processed on-the-fly would significantly contribute to the development of cyber threat prevention techniques. This means that a network testing environment rather than ability to only read in the data files is needed. The components of such a test bed are presented in Fig. 2. First, the reference set is divided into a learning and a testing set. The learning set is used to develop the traffic analyser using machine learning techniques. The testing set may be contaminated by the events from an outside source, e.g. sample testing cases with known expected response. Such testing cases are needed to verify the merits of the forensic techniques. Next, network traffic can be generated based on the contaminated testing set. The volume of network traffic can be scaled to test the performance of detection techniques in parallel with their accuracy.

The primary objective of this work is to propose such a test bed and evaluate it using a realistic test scenario based on spam detection. Such a combination of virtual test bed and data sets based on real network traffic is described in this study. The problem dealt with is spam detection based on NetFlow records, which are both developed and processed on-the-fly. The NetFlow records are produced by a NetFlow probe named nProbe[1] and processed by a newly developed plug-in implementing machine learning algorithms. The role of the plug-in is to extend the NetFlow mail records with the additional attribute meaning spam or correct mail flow. Such extended NetFlow records can be processed locally or by a central collector. The tests of the proposed spam detection technique refer to its accuracy, the performance of the modules needed to produce NetFlow records with the required discriminators and the performance of the spam detection module. Hence, not only is the method accuracy aspect addressed but also the impact of data stream and traffic volume on method performance.

The remainder of this work is organised as follows. First, the creation of reference sets is discussed in Section 2. This is followed by a description of the testing environment outlined in Section 3. The way the combination of environment and realistic data has been used for the end-to-end testing of the spam detection module is discussed in Section 4. This is followed by the conclusions in Section 5.

2 Creation of reference sets and detection models

This section contains assumptions that allow the user to create a realistic reference set. The assumptions are defined as generic rules that can be used to develop the reference set for any network cyber threat.

The presented propositions are verified on a real task, namely the detection of spam, which is presented in Section 4.

2.1 Reference set creation

A *reference set* can be defined as a set of network traffic data having a form and content that allows the detection of one or many categories of events of interest. Such a set is usually a subset of the real network traffic set that is large enough to allow the analysis of the event but significantly reduced compared to the source traffic. Realistic and representative data sets can improve research into detection of anomalies, attack prevention and botnet detection.

[1] http://www.ntop.org/products/nprobe/

However, the creation of reference sets is subject to serious problems including, but not limited to, the collection and storage of large volumes of data, the privacy aspect and the relevance of the collected events. Therefore, only a very limited number of useful reference sets exist [7]. Among notable exceptions, the repository of data sets maintained by the Cooperative Association for Internet Data Analysis can be mentioned. Still, even in this repository, the data sets for individual events are quite limited. For instance, there is only one data set for a DDoS attack, originating from 2007.[2]

The creation of reference sets starts with the selection of observed parameters. For the selected parameters, their values are captured. On the basis of these parameters, the features are calculated. Different values of features describe various events, which can be separated from the background network traffic on the basis of logical rules or classifiers. Thus, the reference set is a set of records described by features that can be used to detect events among background traffic. Individual steps for creating such sets are proposed in the following sections. It is assumed that the reference sets will be defined as sets of NetFlow records. However, reference data sets not based on NetFlow records are also possible.

The reference set can be a sample of typical network traffic that contains some event records captured from the network, or injected by an enrichment module. Moreover, the reference set has to contain sufficient information to create data in a form acceptable for a diagnostic unit. The methods of set creation for different applications are proposed in the following sections.

2.2 Features selection

The problem of monitoring in a multi-service and multi-user environment exhibits a high complexity due to the size of the space to explore. In order to be able to detect events in such a context, the data set to be analysed should be reduced [9]. Therefore, only the most important features should be selected to describe the collected records.

For detection of significant features, it is necessary to first define the range of collected data. Discriminators for use in flow-based classification were proposed among others in [16]. Among nearly 250 features proposed in this study, there are such basic features as packet length, TCP window size, or time to live (TTL).

The method of selection proposed in this work is based on machine learning. To perform an analysis, two distinguishable subsets are necessary. The first subset contains the analysed events and the second consists of the rest of the network traffic. Both subsets are described by the same subset of low-level features. Feature selection creates a subset of features that discriminate events from the background traffic.

The initial set of features is limited to simple statistics such as count, minimum, maximum and average. In the next step, the selection of features is performed by a decision tree [4]. This selection results in a subset of relevant features for use in model construction being created.

Although the tree is not a very complex classifier, its main advantage is the ease of interpretation. Even a tree that involves a large number of splits and nodes can be interpreted by users. Predictions of the tree are based on decision rules. In classification problems, the user can specify misclassification cost. This is an important aspect when events with extremely varied economic costs of misdetections such as spam [17] or DDoS attacks [18] are discussed. In the tree creation process, the Gini coefficient is calculated and used as the measure of discrimination ability for selected features [1]. The features with the best values of the coefficient are included into the model. The hierarchy of features is used in the selection process. The features with a small influence on the model are rejected from the set to minimise the computational overhead related to network traffic processing.

The main disadvantage of the method is the fact that each of the rejected features is evaluated individually, while the rejection of several seemingly irrelevant features can decrease the accuracy of the model. Therefore, the model given by the reduced set of features should be compared against the model created for the whole set of features.

The proposed method of selection yields a set of basic but significant features. The reduction of features to the basic statistics allows a collector to calculate the features relatively fast. The analysis made by the tree replaces the feature set with its subset containing the most significant features.

2.3 Flow collection

Packet header analysis and flow analysis are the main approaches that should be discussed in the context of multi-gigabit stream analysis. Neither of them utilise the information contained in the payload. This fact is very important for data volume reduction, but above all, to respect clients' privacy. Therefore, a hash function can also be used to obfuscate the IP addresses. Moreover, it was proven that data does not need to be stored in some cases, as it can be or even has to be processed in an on-line manner. Examples include the statistical detection of DDoS attacks [14] and solutions developed on field-programmable gate array (FPGA) cards [12].

Both hardware devices and software tools such as nProbe can be used to constantly collect traffic data and emit NetFlow v9 flows towards a specified collector [6]. The set

[2] http://www.caida.org/data/overview/

of collected features is defined by a template. Default templates can be used to capture most of the basic features. Moreover, the collector can also be extended by plug-ins to provide other features.

The size of collected flows is significantly lower than the size of complete frames or even just headers. However, even for a very short period, the collection of data from a broadband remote access server (BRAS) creates multi-gigabyte files. Therefore, only a limited subset of the traffic can be analysed. There are two approaches to creating the subset. In the first approach, the subset is a cross section of the traffic. In this case, a random pool of clients' IP addresses is selected and observed. The pool must maintain a typical proportion between various types of clients (business, individual). In the second approach, the subset is a probe of a typical environment where events occur. Such an environment can be defined by some protocol and ports. These two approaches can be combined.

The filtering rules for the IP pool as well as for the protocol and ports can be described and applied during the collection process in the form of Berkeley Packet Filter, which is supported by nProbe.

2.4 Analysis of probe

A probe collected from the whole traffic should be checked for the presence of the analysed events. This task can be done by supervised learning. Several machine learning techniques can be used, but such classifiers must be trained on a learning set that consists of distinguishable events and background traffic records. The classifier works as a binary separator and separates the events from the rest of the traffic. It is assumed that at least a part of the instances will be incorrectly classified. Such records create a new class called 'other.' The size of the class determines the classifier's confidence level (better classifiers have a smaller other class). After that, the other class is created and a new classifier is trained. The classifier splits the data space into three classes: an event, background traffic and other. The model created by the classifier includes uncertainty about the result.

The classifier trained on a learning set has a tendency to detect the events described by the learning set while the events from the probe may have different characteristics. Therefore, an ensemble of various classifiers trained on different data sets should be used instead. The schema of the probe analysis and the classifier development is shown in Fig. 3.

The classifiers in the ensemble have various accuracies and confidence levels. Therefore, a final classification should not be carried out by an unbalanced voting algorithm. The following method is proposed instead. Each classifier from the ensemble recognises three classes: an event, background traffic and other. The latter class contains all border cases. The classes are labelled by 1, −1 and 0, respectively. The classifier C_i that returns a decision y_i is described by two coefficients. The first one, s_i, is the accuracy of discrimination between the events and the background. The second one, c_i, is a confidence level calculated as a percent of decisions from outside of the other class calculated as:

$$c_i = 1 - \frac{\sum_{y_i=0}}{\sum}. \tag{1}$$

The final classification decision is the sign of a weighted sum given by the formula:

$$y = \operatorname{sgn}\sum_{i=1}^{n} \frac{s_i}{\sum_{j=1}^{n} s_j} \frac{c_i}{\sum_{j=1}^{n} c_j} y_i, \tag{2}$$

where n is the number of classifiers in the ensemble.

The presented analysis method is essential for the whole proposed methodology. Various sources of data are necessary to create a universal probe. On the other hand, the ensemble of classifiers is a tool that can be easily extended to improve the classification quality.

2.5 Enrichment of probe

If the number of records in the probe describing the analysed event is scarce, then new records should be added. As stated before, the probe can be used just as a background for the observed events. Then, the events from a learning set can be added to the probe or the probe already contains interesting events and new records should be generated based on the existing records. Both approaches are described in the following sections.

2.5.1 Contaminations from outside sources

Adding data to the probe brings several problems. Most of all, contaminations from outside sources should have the same set of features as the probe. Otherwise, features that describe contaminations must be transformed to the proper form. This problem can be avoided if the same technique is used to collect data in both cases.

The second problem occurs especially when additional data is collected in the same network. In such a situation, the data may be described with the same source and destination as existing flows. If flows concern different pairs of sources and destinations, where a source and a destination is defined by the IP number and port, then a new set of flows can be simply concatenated with the probe. Otherwise, the features should be recalculated. It is assumed that all features are simple statistics such as count, minimum, maximum and average.

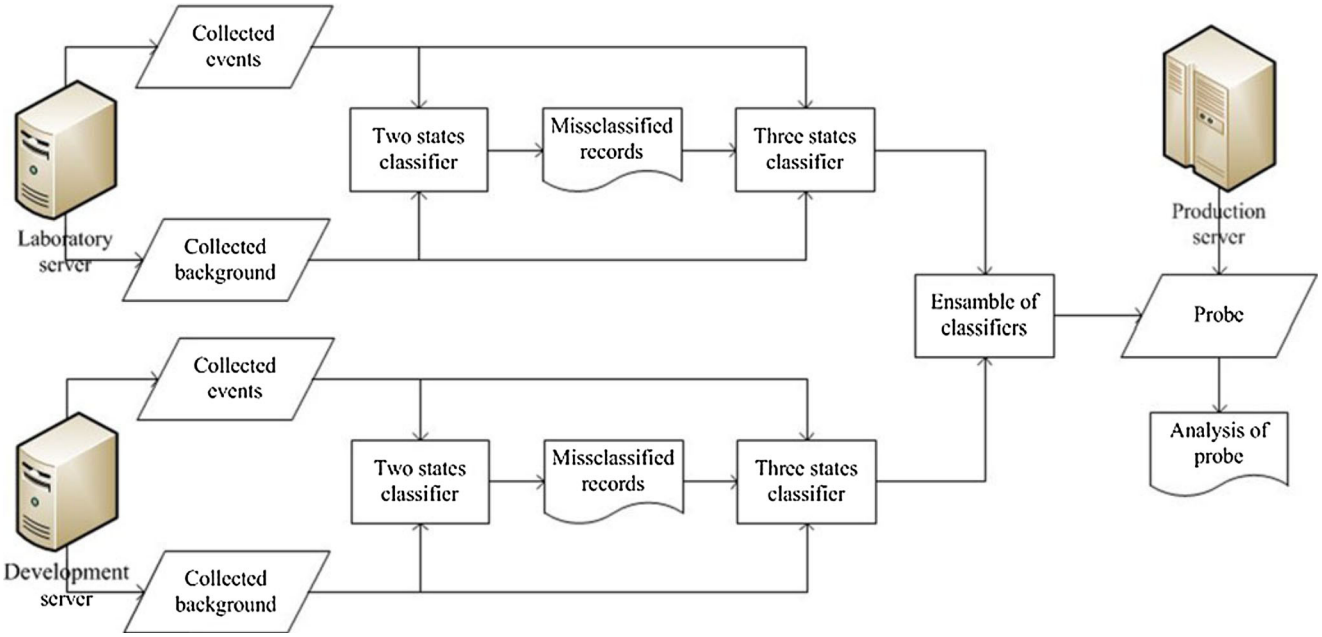

Fig. 3 Analysis of the probe. Data collected from various servers are used to create binary classifiers. Records misclassified by the classifiers create a new class. In turn, multi-class classifiers that split data into ultimate classes, being the event, background traffic and other, are developed. The classifiers are grouped in an ensemble. The ensemble is used to analyse the probe

If in the probe P and in the outside set O two flows with the same source and destination exist, then a new statistic for the feature f is calculated from features f^P and f^O as:

$$\begin{aligned} f_{\max} &= \max\left(f^P_{\max}, f^O_{\max}\right), \\ f_{\min} &= \min\left(f^P_{\min}, f^O_{\min}\right), \\ f_{\text{count}} &= f^P_{\text{count}} + f^O_{\text{count}}, \\ f_{\text{avg}} &= \frac{f^P_{\text{count}} f^P_{\text{avg}} + f^O_{\text{count}} f^O_{\text{avg}}}{f_{\text{count}}}. \end{aligned} \tag{3}$$

New records are added to the probe because the proportion of records which describe the event is relatively small. However, the new records may cover original records and change the characteristic of the event. This can be beneficial if the new records describe the specific aspect of the observed event. Nevertheless, in the situation when records from the probe are the main aim of research, a different methodology should be used. The methodology is described by the following algorithm.

Let R be a subset of the probe that describes the event and the pair $(O, \preceq)$ is a partially ordered set, where O is a set of outside records and r is a partial order relation over a set O. The relation $\preceq$ is defined as

$$\forall_{o_1,o_2 \in O} o_1 \preceq o_2 \equiv \min_{r \in R} d(o_1, r) \leq \min_{r \in R} d(o_2, r), \tag{4}$$

where the function $d(o, r)$ is a distance calculated between the records o and r. Usually, the distance between records is the Euclidean metric

$$d(o, r) = \sqrt{\sum_{i=1}^{n} (o_i - r_i)^2}. \tag{5}$$

The records can be normalised to the rage [0, 1] to level off influence of features. However even then, the distance for binary features is always 0 or 1 and individual features are mostly independent. Therefore, the Manhattan distance can be used instead

$$d(o, r) = \sum_{i=1}^{n} |o_i - r_i|. \tag{6}$$

An application of the reaction $\preceq$ implies that the first records of the list created by the ordered set O have similar equivalents in the set R.

Now, subsequent records from the list can be added to the set until a desired quantity is obtained. An alternative approach assumes that records are added until the value of d is below some threshold.

2.5.2 Contaminations generated by genetic algorithm

Contaminations may also be generated without outside influences using a genetic algorithm. The algorithm creates two new records based on two existing records. Each

record is described as a set of continuous and discrete features. Randomly selected records are transformed into new records by crossover and mutation operators. The fitness of the created record is checked by a classifier that recognises the event. The details are given below.

Each flow is described by a set of continuous $c_1, c_2, \ldots, c_n$ and discrete $d_1, d_2, \ldots, d_m$ features. Therefore, two randomly selected flows from the set R can be described as:

$$r^1 = c_1^1, c_2^1, \ldots, c_n^1, d_1^1, d_2^1, \ldots, d_m^1,$$

$$r^2 = c_1^2, c_2^2, \ldots, c_n^2, d_1^2, d_2^2 \ldots d_m^2. \quad (7)$$

The records r^1 and r^2 can be used to generate new records through genetic operators: crossover and mutation. The crossover operator works separately on continuous and discrete features.

New continuous features are calculated as averages of parents' values:

$$\forall_{i=0}^{n} c_i^a = \frac{c_i^1 + c_i^2}{2}. \quad (8)$$

The same method cannot be applied to discrete features. Therefore, two copies of a flow are created. Both flows have the same continuous features but their discrete features are inherited from different parents. Hence, new records are defined as:

$$r^3 = c_1^a, c_2^a, \ldots, c_n^a, d_1^1, d_2^1, \ldots, d_m^1,$$

$$r^4 = c_1^a, c_2^a, \ldots, c_n^a, d_1^2, d_2^2 \ldots d_m^2. \quad (9)$$

The mutation operator can be used optionally and should be used when the initial number of flows is very small. The operator changes one of the discrete features into its opposite:

$$\exists!_{i \in \{1,\ldots,m\}} d_i = (1 - d_i), \quad (10)$$

where $\exists!$ means that there is one and only one discrete feature changed.

The created records can be considered as the event if and only if the classifier recognises them as the event. Only verified records should be added to the probe R. In other cases, the records should be generated once again. An exception can be made when the probe R is created to verify heuristic methods that should recognise a new form of events. In such a case, additional limitations on the form of created records depend on the type of events.

2.6 Procedure

The assumptions presented in this section create a coherent methodology. The whole method is summarised in Algorithm 1. An input of the algorithm is a labelled set of PCAP and classifiers. As an output, a reference set is created. The set consists of NetFlow records that are labelled by classifiers trained on input data. The set can be extended by additional records.

3 Creation of testing environment

3.1 Introduction

Constructing classifiers detecting anomalies in network traffic involves performing many rigorous tests to evaluate their performance. Setting up a proper testing environment for such a purpose is not a trivial task. First of all, it should make trying out various approaches as easy as possible under laboratory conditions. Secondly, it should allow experiments to be performed on different scales—ranging from simple proof of concept checks performed on single machines to much larger experiments involving e.g. hundreds of computers connected to a local network.

This general concept translates into a number of important requirements from the experimenters' point of view. An ideal testing environment would therefore:

1. Make implementing the developed methods a straightforward task, and thus incorporate a rich programming environment with a variety of robust libraries and easy access to the underlying network traffic,
2. Be easily pluggable into many points of a complicated network,
3. Be adaptable to the desired scale of the test (up to a reasonable extent),
4. Have the ability to introduce anomalies into the network in a controlled fashion,
5. Be robust and easy to set up even on low-end hardware.

Notice that the hardware requirements are rather vague as these are strictly connected to the specific anomaly types and methods being tested and cannot be stated in the general case.

3.2 Proposed solution

The testing environment we propose is a distributed system consisting of probes plugged into key points of some network. Each of these probes monitors the passing traffic and performs computations connected with extracting its features and applying the tested anomaly detection methods.

Algorithm 1 The creation of a reference set

Data:
T_d – the labelled set of training PCAP
C –the set of classifiers
O_f –the outside probe
Result:
R_s // The reference set
begin
  $N = \emptyset$; // The set of NetFlows
  foreach $sd \in T_d$ // sd -- a unique pair (source, destination)
  do
    $n = \{source, destination\}$; // a NetFlow
    foreach $a \in ds$ // a -- an observed attribute
    do
      $s = \emptyset$; // s -- a set of new features
      $s = s \cup count(a)$;
      $s = s \cup avg(a)$;
      $s = s \cup min(a)$;
      $s = s \cup max(a)$;
      $n = n \cup s$;
    end
    $n = n \cup \{sd_e\}$; // sd_e -- the label of source--destination traffic
    $N = N \cup \{n\}$;
  end
  $N_r = tree(N)$; // Use a tree to reduce features
  foreach $C_i \in C$ // Procedures described in 2.4
  do
    Train C_i on N_r to recognise sd_e;
    Create the 'other' class;
    Retrain C_i including the 'other' class;
    Calculate s_i and c_i;
  end
  Calculate R_f – equivalents of N_r for a real, filtered data;
  foreach $n \in R_f$ // n -- a NetFlow
  do
    foreach $C_i \in C$ **do**
      $y_i = recognise(C_i, n)$; // the classification of n by C_i
    end
    $sd_e = calculateLabel(y_1, y_2, \ldots)$; // see eq. (2)
    $n = n \cup \{sd_e\}$;
  end
  if *proportion of labelled records are adequate* **then**
    $R_s = R_f$;
  else
    if *an outside probe* O_f *is given* **then**
      $R_s = R_f \cup O_f$; // Procedure described in 2.5.1
    else
      $R_s = geneticEnrichment(R_f)$; // Procedure described in 2.5.2
    end
  end
  return R_s;
end

In such an approach, the probes are independent so scaling to a larger network simply involves increasing their number. It should also be noted that when the tested detection methods are error prone, robustness may be obtained through redundancy—then a failure of a single probe does not corrupt the entire testing system.

In the case of more complex situations or more computationally expensive methods, such an environment may be extended by dividing the responsibilities and introducing hierarchy. In such a situation, the testing environment consists of interconnected nodes that can be divided into three general types:

1. Probes directly processing ongoing network traffic, extracting the necessary features and performing initial calculations,
2. Analysers aggregating the results obtained from probes and other analysers and performing higher level calculations,
3. Collectors storing the calculation results (e.g. in a database).

The concrete number of node types and their responsibilities of course need to be carefully tailored to the specific anomaly type and methods. The benefits of splitting the computational effort include better scaling to large traffic loads and support for the implementation of sophisticated methods requiring information originating from distinct points of the underlying network.

One of the tools available for network monitoring is nProbe, which has already been mentioned in this article. Its basic use involves monitoring network traffic and generating NetFlow flows which can either be directly collected, exported further or both. Another possibility is that it may become a proxy which receives and transforms the flows originating from other sources. After the transformation, the flows may also be collected directly or exported further. The operations performed while monitoring network traffic and transforming the flows may be extended by custom plug-ins written in the C programming language. As this language has a rich ecosystem consisting of countless tools and libraries, it provides a suitable programming environment for creating efficient implementations of anomaly detection methods. All these properties make nProbe an ideal base for implementing the proposed testing environment.

The possibility of introducing specific anomalies into the network is a crucial element of the testing environment. This can be performed by at least two methods: replaying previously recorded packets and generating packets according to some given flow. The first method is a straightforward task which may be technically performed using for example tcpreplay.[3] However, it requires a PCAP file with the recorded anomaly. It should be noted that care needs to be taken in order to prevent possible legal problems connected with storing the payload and IP addresses.

The second anomaly injection method is somewhat more complicated and based not on raw packets but on flows recorded during the anomaly occurrence. This approach may be easily explained if we notice that a flow r may be interpreted as a value of function Φ which aggregates a sequence of packets ψ, so $r = \Phi(\psi)$.

If it is possible to define a reverse function Φ^{-1} which generates a sequence of packets from the given flow ($\hat{\psi} = \Phi^{-1}(r)$), then, if these packets are injected into the network, the monitoring probe should record the occurrence of flow r.

Such a generated sequence will of course differ from the original one, so we have $\hat{\psi} \neq \psi$. However, as the tested classifiers operate on flows rather than raw traffic, a situation in which the probe records the desired flow r is sufficient for experimental purposes.

The question remains whether it is possible to construct the Φ^{-1} function. This of course depends on the types of features that are present in the flow. In some cases, creating a proper sequence of packets will be trivial, e.g. if we are only interested in the minimal $l_{\min}$ and maximal $l_{\max}$ packet length and the total length of the flow l_{tot}, we simply need to construct a sequence of packets consisting of:

1. a single packet of length $l_{\min}$,
2. if $l_{\max} > l_{\min}$: a single packet of length $l_{\max}$,
3. if $l_{\text{tot}} > l_{\max}$: k packets such that each one has a length in the range $[l_{\min}, l_{\max}]$ and the total length of all the k packets is equal to $l_{\text{tot}} - l_{\max} - l_{\min}$.

However, there exist features for which constructing a reverse function might be very difficult. A good example would be a feature which is the result of using some sort of hash function on the packets. In the case of methods operating on features which are results of simple statistical functions, such a situation will never occur.

Technically, such an injection may be performed using a packet crafting tool such as, e.g. Nemesis[4] or Mausezahn.[5]

4 Use case: spam detection

The methodology described in Section 2.1 was used to create a reference set that contains spam. All steps of the method are presented in the following sections.

[3] http://tcpreplay.synfin.net/

[4] http://nemesis.sourceforge.net/

[5] http://www.perihel.at/sec/mz/

4.1 Features selection

Flow-level parameters selected in [20] as a subset of the set defined in [16] are the base for features selection. The features were calculated for flows collected by Žádník and Michlovský [20]. The authors collected data from the SMTP server hosting mailboxes of the Liberouter[6] project group. The data set contains over 58,000 records described by 64 features and divided into several classes. Among all classes, two describe spam. The first class *dnsbl* contains flows from IP addresses mentioned on DNS black lists.[7] The second class *y_spam* consists of flows that were successfully received and marked as spam by SpamAssassin.[8] In this paper, both classes are considered as a single class *spam*.

The same set of features was used to develop the new set of NetFlow records that was collected at Warsaw University of Technology. The set originates from the mail server Alpha and consists of NetFlow records containing the same collection of features as Žádník's set. The data was collected over one working week. More than 42,000 NetFlow records were collected. Among them, 589 records were labelled as spam. The labelling method was similar to the method used in [20]. SpamAssassin logs were compared with collected flows. The spam detector decision was ascribed to a flow with the same source and a close receipt time.

It was shown in [8] that the set of 64 features proposed in [20] contains partly redundant features and the dimensionality of the data can be reduced. Therefore, feature selection should be considered to eliminate redundant and irrelevant features.

Discrimination rules that separate spam from the rest of the traffic were developed for both created sets. This task was performed using a C&RT tree [4], which creates clear decision rules. A classification tree selects features because of the Gini coefficient [1], which is a measure of statistical dispersion. The feature significance is computed by summing over all nodes in the tree the drop in node impurity. The results are expressed relative to the largest sum found over all predictors where the largest sum is assigned 100 points. The details of this method are given in [4].

The accuracy of both classifiers was about 97 %. However, the tree structures were different. Therefore, various features were selected as the most significant. The final features set should not be based only on features selected as major by both classifiers because such a set would be very limited. Instead, the set can be extended by the features selected by only one classifier. It is worth noting here that different features may provide the same information. Hence, inevitably different feature sets may be used by different classifiers.

The most significant features are collected in Table 1. Features are calculated separately for both directions. Therefore, the final set consists of 24 features.

4.2 Capture NetFlow records and features calculation

NetFlow records were collected by nProbe. This tool allows the user to define a template of a NetFlow record. The template, which was used in the case of spam, consists of IPv4 data, information about the number of bytes and packets, flags and others. Traffic was filtered using the BPC *Berkeley Packet Filter* rule that limits collected data to the mail traffic. As a result, 176,446 records were created. The total size of records was 17.5 MB.

The collected records should be rewritten to achieve a form similar to the set of the most important features that is presented in Table 1 . Several features such as the count of packets and the minimum and maximum lengths of packets can be calculated directly from a built-in nProbe template. The features based on flags such as the count or average length of packet having a given flag cannot be calculated from the template. Bit information about the flags set is calculated instead. Eventually, the created set of features consists of simple statistics calculated for packet length and binary information about flags presence.

4.3 Probe analysis

Two classifiers were used to separate the traffic into three classes: spam, background traffic and others. The first one was a support vector machine (SVM) [21]. The second one was a random forest [4]. Both techniques were trained on records from Žádník's and Alpha sets separately. The sets consist of two classes, i.e. spam and background traffic. The

Table 1 The most significant features describing spam

Significance	Name	Description
100.0	slas	Average length of packets having the ACK flag
99.5	spl	Average packet length
91.0	slps	Average length of packets having the PUSH flag
90.5	maxpl	Maximum packet length
84.0	maxtw	Maximum TCP window size
82.5	spps	Count of packets having the PUSH flag
73.0	stw	Average TCP window size
68.0	mintw	Minimum TCP window size
67.5	maxttl	Maximum TTL
67.0	sp	Packets count
66.5	minttl	Minimum TTL
66.5	sttl	Average TTL

[6] http://liberouter.org/

[7] http://cbl/abuseat/org/

[8] http://spamassassin.apache.org/

third class 'other' was created from records misclassified by the binary classifiers. The final classifiers were trained on the three classes.

The obtained results are at least as good as those presented in other works. However, it is hard to compare the obtained results directly. For example, in the work [13], the machine learning techniques gave 93 % accuracy, but the classified set was different form the one discussed in our works. In the work [20], the same set of features was used for the same records (Žádník's set) but the classifiers recognised five classes. The precision was about 96 % in this case. Finally, in the work [8], Žádník's set was separated into three classes: spam, correct and others. The best result obtained by an MLP network was 96 %. However, in the last article, the main aim of the work was the dimensionality reduction rather than classifier tuning.

The probe collected from BRAS was analysed by ensembles of classifiers based on NetFlow features. The analysis shows that the observed event spam is almost imperceptible in the probe. The share of spam in the probe is less than 0.5 %. If the probe were to be used as a spam reference set, contamination should be added.

4.4 Created reference sets

Apart from the collected probe, two additional reference sets were created. In both sets, the number of spam records was increased by 19,000. That gives about 10 % of spam in the analysed probe, whereas the participation of spam was imperceptible in the original set.

The first set was created by adding records from the outside set (the approach was described in Section 2.5.1). The records came from different sources and thus a simple concatenation was enough to merge sets. The contamination changed the characteristic of spam to a significant degree.

The contamination of the second set was generated by a genetic algorithm, based on the approach described in Section 2.5.2. The new records were generated on the basis of existing ones and the characteristic of spam changed minimally.

The first created set can be used as a training set for methods that should recognise the given type of spam among typical traffic. The second set can be used as a training set for methods that should detect any kind of spam in the analysed environment.

In the spam detection case, the a priori probability is sometimes too small to create a valid classifier. This problem can be solved by the creation of a balanced learning set. This can be done by reduction of records in a dominant class or by adding new spam records.

Table 2 presents classification results for SVM classifiers trained on various data sets including the original set. An RBF kernel was used with $\gamma = 0.8$ and $C \in [1, 10]$. The evaluation of classification on the sets was performed by a 10-fold cross-validation.

The a priori classification probability for spam in the original probe was too small to create a classifier that detects spam. However, the spam detection can be improved if a reference set is used.

The classifier trained on the additional outside set detects over half of spam records, but has a higher false positive ratio. However, false alarms may be less expensive than missed events.

Records from the outside set may have different characteristics than spam records from the original probe. Therefore, one more test has been performed on the generated records.

The generated records have the same characteristics as records from the probe. The classifier trained on this set can detect spam similar to the spam given by the probe. Only 1 % of spam is detected but it may be assumed that the recognition of spam can be improved by increasing the number of generated records.

4.5 Testing environment and experimental results

In order to apply the developed spam detection methods to real network traffic, a testing environment was prepared following the concepts presented in Section 3. At the core of this environment lies an nProbe plug-in which classifies the flows gathered from the underlying network traffic using the provided classifier.

When the probe exports a finished flow, the plug-in effectively becomes active and performs a number of tasks:

Table 2 Classification results for various reference sets

Set	Class	Count	Correct	Incorrect	Correct %	Incorrect %
Original	Background	43,875	43,875	0	100	0
	Spam	239	0	239	0	100
Original	Background	43,876	42,915	961	98	2
+ Outside	Spam	4,319	1,983	2,336	45	54
Original	Background	43,879	43,879	0	100	0
+ Generated	Spam	4,985	54	4,931	1	99

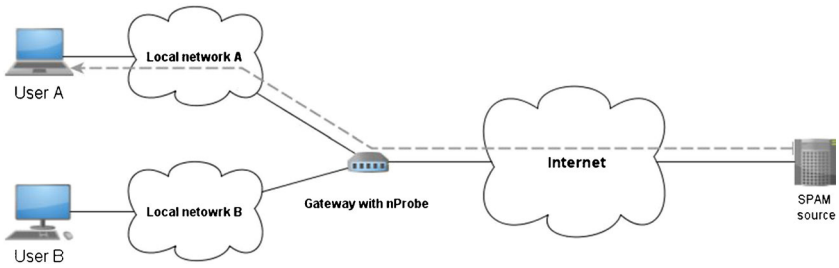

Fig. 4 An example network setup: two local networks connecting to the internet through a gateway with nProbe

1. It calculates the flow features required by the classifier,
2. It transforms the calculated features into data structures acceptable for a given classifier,
3. It runs the classifier and collects the resulting class,
4. It appends both the calculated features and the resulting class to the exported flow.

Thus, we are able to apply these models under real-world conditions. In the case of the presented results, the classifier was a random forest model prepared during our experiments.

A typical use case for the plug-in is monitoring networks for spam in situations similar to the one depicted in Fig. 4.

In this example, there are two networks connected through a gateway that constantly monitors the traffic using nProbe with the spam plug-in activated. If some source host starts sending spam to the computers in local network A, the plug-in will be able to detect this procedure and the host may be blocked.

To gain an insight into the computational performance of the presented environment, load tests were performed. The purpose of these experiments was to determine whether using the spam plug-in has a significant influence on nProbe's throughput. To carry out network tests, three general approaches may be identified: simulation, live experimentation and emulation [15]. Simulation is computationally extensive and is most suitable for simplifying problems, focusing on their specific aspects or when hardware and firmware modifications are required [3]. Therefore, it is not feasible for load testing purposes. As (in comparison with live testing) emulation provides better control over the network elements, the tests were performed in an emulated virtual environment.

The experiments were conducted using a simple network consisting of only two hosts. The first one was injecting traffic while the second host was a server monitoring the network using nProbe and measuring the times needed to prepare NetFlow data. The computational effort required during such data preparation is of course directly connected to the list of exported fields. During the load tests, we focused on estimating the extent to which using the spam plug-in influences this procedure, thus the three different lists described in Table 3 were used. For each list, a series of independent experiments with increasing traffic volumes were performed—the lower and upper traffic levels were approximately 10,000 and 20,000 packets per second.

One important aspect of the experiment is connected with injecting the network traffic. As the spam plug-in only performs computations once per flow and the purpose of the load tests was to measure its overhead, there should be a large number of distinct flows generated during these tests rather than only a large number of packets. Thus, the traffic injected into the network consisted of short 10-packet flows generated from a base sample of 10,000 flows by randomising the host IP addresses. The base sample originated from the already mentioned Alpha data set and provided sufficient packet variety for load testing purposes.

In order to verify the claims about the low hardware requirements of nProbe, the experiments were performed on hardware with computational performance which is low by modern standards. For that purpose, nProbe was installed on a virtual server with significantly limited CPU speed, which in terms of floating point operations per second (FLOPs) roughly corresponded to Intel Pentium III @750 MHz processors. This can be verified by analysing Fig. 5, which

Table 3 NetFlow fields used during load testing experiment setups S1,S2 and S3

S1	S2	S3	Field code	Description
✓	✓	✓	IP4_SRC_ADDR	Source host IP4 address
✓	✓	✓	L4_SRC_PORT	Source host port
✓	✓	✓	IP4_DST_ADDR	Destination host IP4 address
✓	✓	✓	L4_DST_PORT	Destination host port
✓	✓	✓	PROTOCOL	Communication protocol
	✓	✓	spam_FEATURES	Input features for spam classifiers
		✓	spam	Random forest classification result

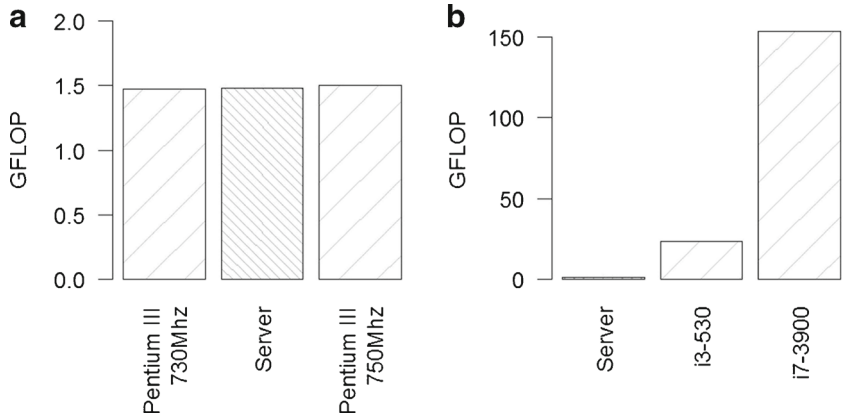

Fig. 5 The computational capabilities of the server in comparison to legacy (**a**) and modern (**b**) CPUs. The comparison is based on the number of GFLOPs obtained by the server in the linpack benchmark (e.g [22]) which is compared to values stated in Intel's microprocessor export compliance metrics. It may be noticed that the server's performance is very poor by modern standards

presents a comparison of the GFLOPs performance metric for various CPU types. Although we were not able to slow down other characteristics of the hardware used, e.g. the memory access latency, we believe that the approach used is sufficient to emulate a network probe based on low-end hardware.

The resulting data from the experiments was collected and preprocessed. The preprocessing consisted in removing the lower and upper 10 percentiles of the measured times. This was necessary as measuring computation times in nanosecond resolution is very sensitive to e.g. background operating system operations which may significantly distort the results. The resulting data was used to calculate the mean export data generating times for different experiment runs as presented in Fig. 6. The differences between the three setups are clear, although it may be noticed that applying the spam classifier only increases the computations by roughly 9,000 ns, which is a satisfying result (this is also summarised in Table 4). The overall loss of throughput may be evaluated by analysing the occurrences of packet dropping during the experiments. It should be noted that the

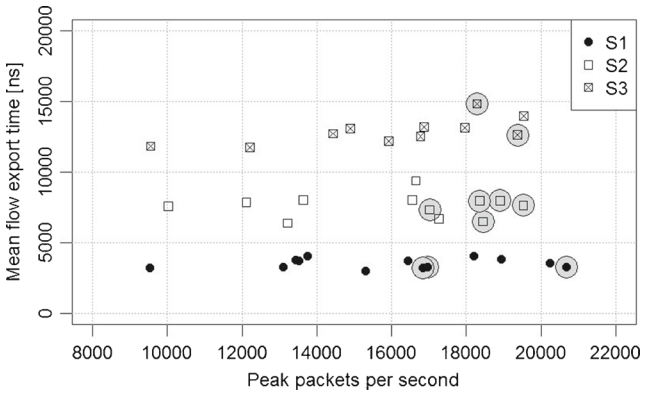

Fig. 6 Mean times required to prepare the flow data measured during the experiments. The three plotted series reflect the S1,S2,S3 setups. The points corresponding to runs with observed dropped packets are surrounded with a grey circle

number of packets per second influences the general computational load of nProbe. However, the effort required to prepare features for the spam plug-in and perform classification is connected with the number of flows per second rather than the number of packets per second. As already mentioned, during the experiments, short flows were used in order to put the focus on overloading nProbe due to exporting rather than capturing packets.

The results obtained suggest that when the traffic load is greater than approximately 16,000 packets per second, the nProbe server becomes overloaded and thus at times drops packets. Analysing the data, there is however no clear pattern suggesting that applying the spam plug-in significantly increases the overload level. Therefore, we may conclude that it does not cause a significant decrease in the overall nProbe throughput, so the upper performance limits will be rather imposed by nProbe itself, not the proposed plug-in.

In order to gain insight into how the obtained results correspond to real production environment conditions, a 23-h anonymised NetFlow sample originating from one of Orange's BRAS was analysed. During the analysis, flows connected with sending emails were identified and used to estimate the magnitude of this kind of traffic. There were approximately three such flows per second and their packet rate, estimated using a 1-s moving average window, was $71 \pm 386[pps]$, which is significantly less than the rate which was analysed during the low-end hardware experiments. This means that the entire email traffic from a single BRAS could be handled by the legacy hardware used during the experiments.

Table 4 Times (in nanoseconds) required to prepare the flow data measured during the load testing experiments

	S1 times [ns]	S2 times [ns]	S3 times [ns]
Mean	4,157	8,u445	15,051
Median	3,594	7,597	12,684
Std. dev.	2,230	4,263	7,558

Presented approximation is simple but adequate for the discussed case. More complex model can be used in the future (cf [10]).

The question arises: what is the upper traffic limit for nProbe with the spam plug-in? As the maximal nProbe[9] throughput is not significantly limited by the spam plug-in, a rough performance estimate may be based on data provided with this software and used to answer such a question. According to the performance data published by ntop[10] when run on dual core hardware, the probe is easily capable of handling traffic above $1,000$ $[Kpps]$ without dropping packets. This means that it should also be possible to analyse traffic of similar magnitude in real time using nProbe with a spam plug-in installed on a modern multi-core machine with an appropriate network interface. Alternatively, a hardware solution can be used. The preliminary analysis suggests that the proposed method or its simpler variation can be implemented on FPGA cards [2].

5 Conclusions

Development of representative data sets and evaluation of threat detection techniques are considered the major challenges in network data analysis. Hence, a way of developing a combination of reference data sets and a testing environment has been proposed. First, the need for reference data sets and the benefits of reference data sets composed of NetFlow records were discussed. This was followed by a description of how the network traffic capturing and transformation process can be organised. Equally importantly, its relation to the use of machine learning methods has been addressed.

The approach proposed in this study was validated using spam detection as a test case. Real data sets composed of mail records were developed and reduced. Next, they were used to generate network traffic of requested volumes to test both the accuracy and throughput of the spam detection system. Importantly, the tests were made with data based on real network traffic. Moreover, mechanisms for capturing new network traffic and converting it into a useful set of relevant features were proposed. Finally, an end-to-end testing strategy was proposed and performed. This took into account the feasibility of the entire threat detection process including traffic capturing, calculation of flow features and analysis by the proposed threat detection algorithm.

In the future, other threats such as different categories of DDoS attacks will be analysed. This will involve data set creation and development of detection techniques. The experience gained from the analysis of new problems will be used to augment the proposed strategy for end-to-end threat detection testing.

References

1. Behera G (2011) Privacy preserving C4.5 using Gini index. In: 2011 2nd national conference on emerging trends and applications in computer science (NCETACS), pp 1–4
2. Borowik G, Luba T, Falkowski B (2009) Logic synthesis method for pattern matching circuits implementation in FPGA with embedded memories. In: 12th international symposium on design and diagnostics of electronic circuits systems, 2009. DDECS '09, pp 230–233. doi:10.1109/DDECS.2009.5012135
3. Boutaba R, Polyrakis A, Casani A (2004) Active networks as a developing and testing environment for network protocols. Ann Telecommun 59:505–524
4. Breiman L, Friedman J, Olshen R, Stone C (1984) Classification and regression trees. Wadsworth and Brooks, Monterey
5. Claffy K (2013) The 5th workshop on Active Internet Measurements (AIMS-5) report. Tech. rep., Cooperative Association for Internet Data Analysis (CAIDA)
6. Deri L (2003) nProbe: an open source NetFlow probe for gigabit networks. In: Proceedings of Terena TNC 2003
7. Fomenkov M, Claffy K (2011) Internet measurement data management challenges. In: Workshop on research data lifecycle management. Princeton
8. Grzenda M (2012) Towards the reduction of data used for the classification of network flows. In: Proceedings of the 7th international conference on hybrid artificial intelligent systems—volume Part II, HAIS'12. Springer, Berlin, pp 68–77
9. Guyard F, Beker S (2010) Towards real-time anomalies monitoring for QoE indicators. Ann Telecommun 65:59–71
10. Jašek R, Szmit A, Szmit M (2013) Usage of modern exponential-smoothing models in network traffic modelling. In: Zelinka I, Chen G, Rössler OE, Snasel V, Abraham A (eds) Nostradamus 2013: prediction, modeling and analysis of complex systems, advances in intelligent systems and computing, vol 210. Springer International Publishing, pp 435–444
11. Karnouskos S (2004) Community aware network security and a DDoS response system. Ann Telecommun 59:525–542
12. Kobiersky P, Korenek J, Polcak L (2009) Packet header analysis and field extraction for multigigabit networks. In: 12th international symposium on design and diagnostics of electronic circuits systems, 2009. DDECS '09, pp 96–101
13. Lakshmi RD, Radha N (2010) Spam classification using supervised learning technique. In: Proceedings of the 1st Amrita ACM-W celebration on women in computing in India, A2CWiC '10. ACM, New York, pp 66:1–66:4

[9] In the tests, nProbe version 6.9.4 was used

[10] http://www.ntop.org/nprobe/tuning-nprobe-6–4–scalability–and–performance/

14. Limwiwatkul L, Rungsawang A (2004) Distributed denial of service detection using TCP/IP header and traffic measurement analysis. In: IEEE international symposium on communications and information technology, 2004. ISCIT 2004, vol 1, pp 605–610
15. Lochin E, Prennou T, Dairaine L (2012) When should I use network emulation? Ann Telecommun 67:247–255
16. Moore A, Crogan M, Moore AW, Mary Q, Zuev D, Zuev D, Crogan ML (2005) Discriminators for use in flow-based classification. Tech. rep., Queen Mary, University of London
17. Schryen G (2007) Anti-spam measures—analysis and design. Springer, Berlin
18. Segura V, Lahuerta J (2010) Modeling the economic incentives of DDoS attacks: femtocell case study. In: Moore T, Pym D, Ioannidis C (eds) Economics of information security and privacy. Springer, Berlin, pp 107–119
19. Tavallaee M, Stakhanova N, Ghorbani A (2010) Toward credible evaluation of anomaly-based intrusion-detection methods. IEEE Trans Syst Man Cybern Part C Appl Rev 40(5):516–524
20. Žádník M, Michlovský Z (2009) Is spam visible in flow-level statistics? Tech. rep. CESNET
21. Vapnik V (1998) Statistical learning theory. Wiley, New York
22. Wu G, Dou Y, Lei Y, Zhou J, Wang M, Jiang J (2009) A fine-grained pipelined implementation of the linpack benchmark on FPGAs. In: 17th IEEE symposium on field programmable custom computing machines, 2009. FCCM '09, pp 183–190
23. Zargar S, Joshi J, Tipper D (2013) A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks. IEEE Commun Surv Tutor 15:2046–2069. doi:10.1109/SURV.2013.031413.00127