

Table of contents

Table of tables.....	3
Table of figures.....	4
1 Introduction.....	5
1.1.1 Reasons for choosing machine learning algorithms as classifiers.....	5
1.1.2 Reasons for choosing Secure Shell as the application to classify.....	6
1.1.3 Reasons for researching packet-based classifiers.....	7
1.1.4 Our research outline.....	7
2 Background.....	9
2.1 Support Vector Machine (SVM).....	9
2.1.1 SVMs and outliers, balancing error with margin.....	11
2.1.2 Non-linearly-separable data.....	13
2.1.3 Multi-class classification with SVM.....	15
2.1.4 Properties of Linear-SVM that we are trying to benefit from.....	15
2.2 Repeated Incremental Pruning to Produce Error Reduction (RIPPER).....	16
2.2.1 How does RIPPER derive its rule set?.....	16
2.2.2 Multi-class classification with RIPPER.....	18
2.2.3 Properties of RIPPER that we are trying to benefit from.....	18
2.3 Adaptive Boosting with Decision stumps (AdaBoost).....	20
2.3.1 Decision stumps.....	20
2.3.2 Building a classifier from voted combinations of weak classifiers.....	20
2.3.3 Multi-class classification with AdaBoost and Decision stumps.....	22
2.3.4 Properties of AdaBoost that we are trying to benefit from.....	22
3 Methodology.....	24
3.1 Data collection.....	24
3.2 Feature selection.....	27
3.2.1 Why did we not choose all fields in the headers?.....	33
3.3 Data preparation.....	34
3.3.1 Calculating inter-arrival time.....	34
3.3.2 Calculating payload length.....	35
3.3.3 Class labeling for training and test validation.....	35
3.3.4 Final data preparation steps.....	36
4 Experiments.....	37
4.1 Phase 1: Differentiating ssh from other applications.....	37
4.1.1 Experiment 1: Comparing performance of Linear-SVM, AdaBoost with Decision stumps, RIPPER and J48 on public data sets.....	37
4.1.2 Experiment 2: Comparing performance of Linear-SVM, AdaBoost with Decision stumps, RIPPER and J48 on private data sets.....	40
4.1.3 Experiment 3: Effect of reducing proportion of ssh in private data sets on the performance of SVM.....	43
4.1.4 Experiment 4: Effect of eliminating port numbers on the performance of RIPPER.....	46
4.2 Phase 2: Differentiating different ssh services.....	48
4.2.1 Experiment 1: Comparing classifier's performance for differentiating ssh services.....	48

4.2.2	Experiment 2: Effect of reducing features on the performance of RIPPER in differentiating ssh services.....	55
5	Future work.....	60
6	Conclusion	60
7	References.....	62
	Appendix A.....	64
	Appendix B.....	65

Table of tables

Table 1 – Computational costs associated with training with linear and non-linear SVMs on data-sample1, a data set of 10,000 instances	14
Table 2 – Properties of public data sets used	25
Table 3 – Proportions of each network application in private data sets	27
Table 4 – Precision and recall values of the different classifiers when trained and tested on different public data sets	39
Table 5 – Student’s t-tests assuming unequal variances of precision values of classifiers on public data sets	39
Table 6 – Student’s t-tests assuming unequal variances of recall values of classifiers on public data sets	39
Table 7 – Precision and Recall (of classifying ssh) values of classifiers on private data sets	41
Table 8 – Student’s t test, assuming unequal variance for precision values of classifiers on private data sets	41
Table 9 – Student’s t test assuming unequal variance for recall values of classifiers on private data sets	42
Table 10 – Precision values of different classifiers as the proportion of ssh decreases, t-tests of whether effect of proportion reduction on classifiers is statistically significant or not	44
Table 11 - Recall values of different classifiers as the proportion of ssh decreases, t-tests of whether effect of proportion reduction on classifiers is statistically significant or not	45
Table 12 – Percentage of correctly identified instances, Precision, Recall and F-measure values for detecting ssh with RIPPER in data sets that do not have port numbers.....	47
Table 13 – The precision and recall values of the classifiers for each ssh service	50
Table 14 – Average F-measure of all classifiers for each ssh service	50
Table 15 – t-value comparing each classifier based on the F-measures for each service, a one-tail critical t-value at significance level 0.05 is 1.746	51
Table 16 – Percentage of instances correctly identified by the different classifiers.....	52
Table 17 – Precision, recall and F-measure values of RIPPER with different features ...	57
Table 18 – Percentage of instances correctly identified by RIPPER when using different features	57

Table of figures

Figure 1 – Maximum-margin hyperplane and support vectors.....	10
Figure 2 – Effect of outliers on maximum-margin hyperplane	12
Figure 3 – Effect of introducing error terms in to the SVM model	13
Figure 4 – An increase in dimension leading to linearly separable data	14
Figure 5 – Simulating network application on local network and collecting data.....	26
Figure 6 – Using servers external to local network for simulating application behavior .	26
Figure 7 – IP header	27
Figure 8 – TCP header	28
Figure 9 – UDP header	28
Figure 10 – Snapshot from a sample ARFF file used in our experiments.....	34

1 Introduction

Our research goal is to build accurate machine learning classifiers that derive appropriate network traffic patterns required for classifying Secure Shell and the services provided by it using only packet level information.

Our research goal requires us to reason our choice of machine learning algorithms as classifiers, our choice of ‘Secure shell’ as an application to classify, our choice of building packet-based and not session-based classifiers. I’ll explain our reasons and provide a brief description of our research approach in this section. This report details the machine learning algorithms we decided to use in the Background section. It then explains our methodology for data collection and preparation. Finally the report will describe all experiments we carried out and draw conclusions based on our results and discuss the potential future work.

1.1.1 Reasons for choosing machine learning algorithms as classifiers

A valid approach for building network traffic classifiers is to manually examine network traffic and build rule based systems with the help of an expert. For example, a well-known rule for classifying secure shell or ssh traffic is to check the port numbers. Traffic on port 22 is usually ssh traffic. More rules could be laid down for the payload section. For example, if the first packets of a session consist of names of encryption algorithms such as ‘diffie-hellman,’ the session is likely to be an ssh encrypted session. An expert would produce accurate and succinct rules to classify ssh. However, this process is time consuming and has several drawbacks that machine learning algorithms overcome:

- Machine learning algorithms are not vulnerable to protocol changes. If an expert produces a complete set of rules to describe the exact behavior of an application, it is easy to hide from a monitoring system by not following some of the classifying rules. For example, a user could change the port, ssh is running on, from port 22 to a different port.
- Machine learning algorithms do not limit themselves to application properties. They also learn network behavior. This is very beneficial especially if the monitoring system is to be run on a particular network. The system could learn rules to identify the location of ssh servers on the network and usage statistics that allow the classifier to boost its classifying accuracy.

1.1.2 Reasons for choosing Secure Shell as the application to classify

By looking at port-based percentages of network traffic applications, SSH occupies a very small proportion of network traffic, usually 1%. This makes it a challenge to train classifiers to detect it because a very high accuracy (99%) is achieved on testing sets even if the classifier does not detect a single ssh packet. SSH encrypts data, this means that we are limited to IP and TCP headers for information, except for the first few non-encrypted packets that determine the encryption algorithm to be used. Encrypting other application packets with ssh (tunneling) is a common technique to bypass firewalls. Using machine learning to not only detect ssh but also the type of service it is providing for example tunneling, file transfer or remote login is an issue we want to investigate as well.

1.1.3 Reasons for researching packet-based classifiers

The two advantages of packet by packet classifiers are:

- We could classify within a packet the type of application we are running.
- The number of features available with packets are limited, leading to simpler classifiers

However, we are planning to examine session-based classifiers in detail in the future. We also propose a temporal model for session based classifiers in our future work section.

1.1.4 Our research outline

Determining which machine learning algorithms is difficult. Other research [7, 12, 19] have used Hidden Markov Models, AdaBoost and Bayesian networks for traffic classification. We want to examine several genres of machine learning algorithms. We identified the following genres and representative algorithms:

- Function classifiers: Support Vector Machines work by finding a mathematical function that describes a hyperplane separating classes in any dimensional space.
- Boosting classifiers: AdaBoost combine several weak classifiers into a complex classifier that weights the decision of each weak classifier based on training errors, hence it iteratively boosts the performance of the classifier it produces
- Rule based classifier: RIPPER produces rules that describe the training data. It avoids over fitting by ensuring the size of the rules and rule set is bound by a theoretical measure, known as the minimum description length.

In addition we use a tree based classifier solely for producing a graphical model of our training data and rules that describe it. We use J48, a java-based variant of C5.0 for this purpose.

We split our research into two phases: phase 1 involves classifying ssh from all other traffic data sets, phase 2 involves classifying services within ssh.

The following section explains each machine learning algorithm and the properties we expect to benefit from using that machine learning algorithm.

2 Background

2.1 Support Vector Machine (SVM)

In the binary classification problem our goal is to split data into two classes. Our data is represented by a vector of n attributes or n features. Our training data consists of several *feature vectors* in \mathfrak{R}^n labeled by the class they belong to. We will use data point and feature vector interchangeably. Assuming our data is linearly separable; we could find a *hyperplane* that separates our feature vectors. A support vector machine finds a hyperplane that fulfills the following property:

The hyperplane separates the feature vectors with maximum distance between the two classes.

This distance is known as the *margin* and the hyperplane is called the optimal or *maximum-margin* hyperplane. The feature vectors from which the distance to the hyperplane is measured, or the vectors at either side of the margin, are known as the *support vectors*, hence the name ‘support vector machines.’ Figure 1 illustrates the hyperplane separating the two classes in a 2-dimensional space and the support vectors that define the margin.

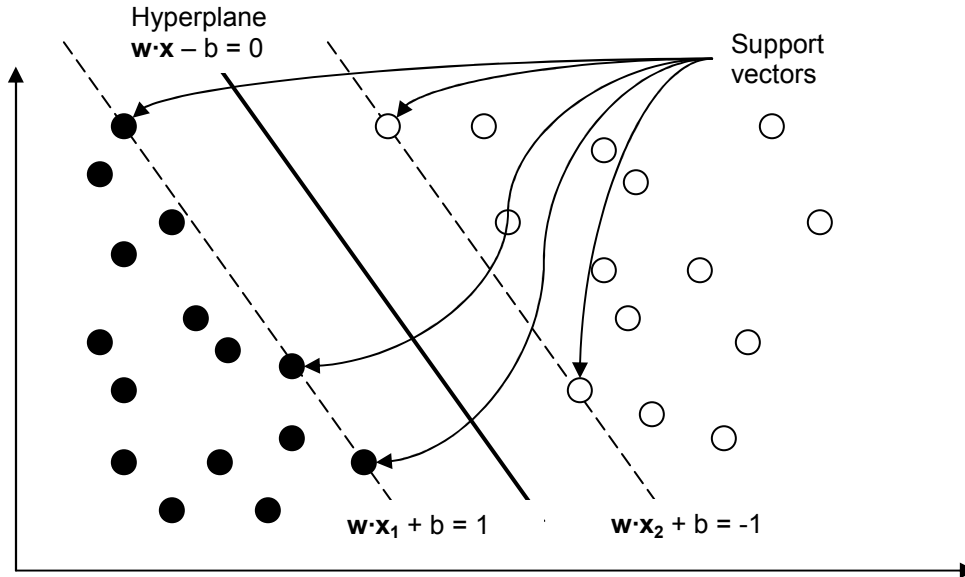


Figure 1 – Maximum-margin hyperplane and support vectors

Training the SVM with linearly separable data is simple. Given a set of feature vectors and their class labels as -1 and 1, we need to find a maximum-margin hyperplane with the following equation, such that it separates the data points with different classes.

$$w_1x_1 + w_2x_2 + \dots + w_nx_n = -b \text{ Or } w_1x_1 + w_2x_2 + \dots + w_nx_n + b = 0$$

Representing our weights as a weight vector, the above equation is the dot-product of 2 vectors. The hyperplane is now defined as $\vec{w} \cdot \vec{x} + b = 0$. Adjusting the values of w and b , we could ensure that the data points on the margin boundaries or the support vectors are located at the hyperplanes $\vec{w} \cdot \vec{x}_1 + b = 1$ and $\vec{w} \cdot \vec{x}_2 + b = -1$. This

ensures that the width of the margin is $\frac{2}{\|\vec{w}\|}$.

$$\begin{aligned} \vec{w} \cdot \vec{x}_1 + b &= 1 \\ \vec{w} \cdot \vec{x}_2 + b &= -1 \\ \Rightarrow \vec{w} \cdot (\vec{x}_1 - \vec{x}_2) &= 2 \\ \Rightarrow \frac{\vec{w}}{\|\vec{w}\|} \cdot (\vec{x}_1 - \vec{x}_2) &= \frac{2}{\|\vec{w}\|} \end{aligned}$$

Therefore, the classification problem reduces to minimizing $\|\vec{w}\|$. An important property is preserved by SVMs: **all data points except the support vectors do not affect the margin**. For any feature vector x_i , we are guaranteed, that it does not lie between our support vectors or in the opposing-class space. A class labeling, c , is easily determined for any data point.

$$\begin{aligned}\vec{w} \bullet \vec{x}_i + b &\geq 1 \\ \vec{w} \bullet \vec{x}_i + b &\leq -1 \\ \Rightarrow c_i(\vec{w} \bullet \vec{x}_i + b) &\geq 1\end{aligned}$$

Where the class label c_i is defined as follows:

$$c_i \begin{cases} 1 & \text{if } \vec{w} \bullet \vec{x}_i + b \geq 1 \\ -1 & \text{if } \vec{w} \bullet \vec{x}_i + b \leq -1 \end{cases}$$

We now minimize $\|\vec{w}\|$ subject to the constraint $c_i(\vec{w} \bullet \vec{x}_i + b) \geq 1$. This optimization problem is called the Quadratic Programming (QP) optimization problem. In certain cases, the QP is an NP-hard problem.

Weka [18], the tool whose SVM implementation we used, reports the values in the weight vector and the intercept b .

There are two problems with the SVM classifier defined so far. (i) It is incapable of dealing with outliers that skew our support vectors. (ii) What if we have linearly inseparable data?

2.1.1 SVMs and outliers, balancing error with margin

Sometimes finding the maximum-margin hyperplane may not lead to the best classifier. This is due to the presence of outliers in data that result from rare cases or errors in the data collection process. This is likely in network traffic data collection. A router, for example, may drop certain packets due to queue overflow. In Figure 2, the

thick line represents a hyperplane preferable to the dotted line. The dotted line hyperplane does not take into account the possibility of error in data and this leads to a classifier that could inaccurately classify new instances of data.

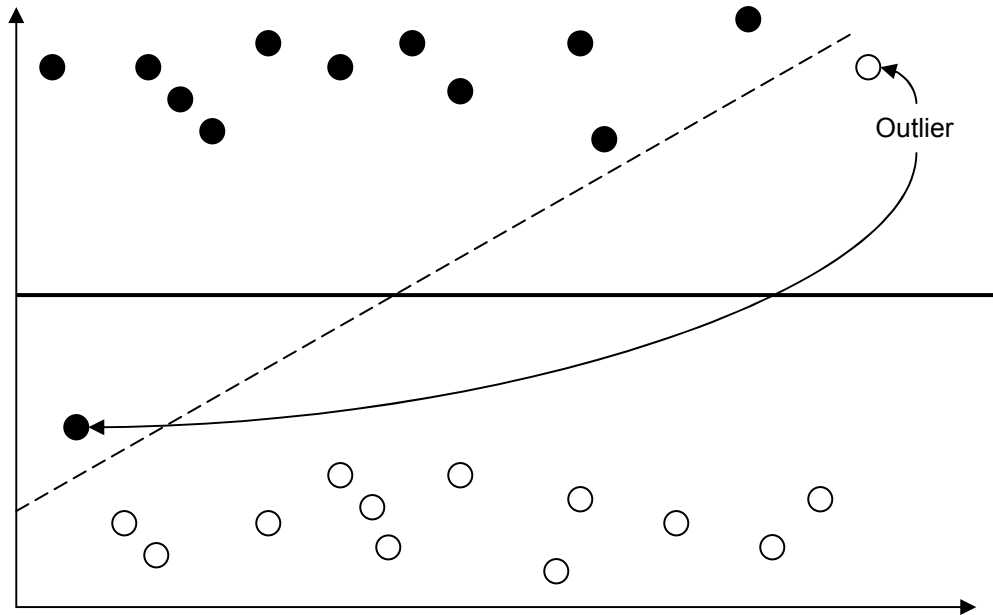


Figure 2 – Effect of outliers on maximum-margin hyperplane

To overcome this problem, a slack variable representing error is introduced into the optimization problem of minimizing $\|\vec{w}\|$. The slack variable allows training data points to lie in the margin defined by support vectors and on opposite sides of the separating hyperplane, i.e. a data point of one class classified as the other class.

$$\vec{w} \bullet \vec{x}_i + b \geq 1 - e_i$$

$$\vec{w} \bullet \vec{x}_i + b \leq 1 + e_i$$

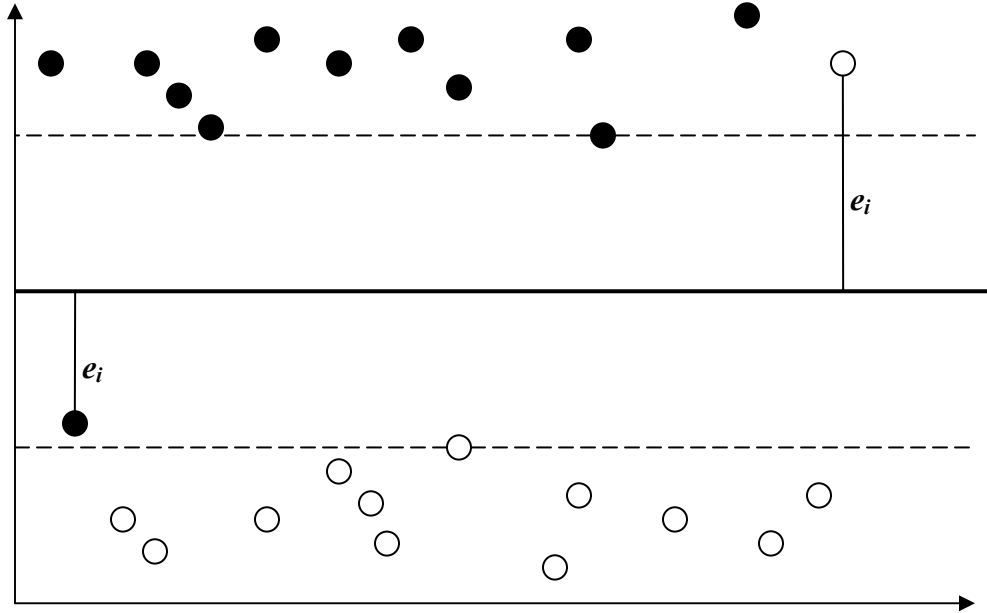


Figure 3 – Effect of introducing error terms in to the SVM model

The number of training errors is bounded by $\sum_i e_i$. Our optimization problem now

involves not only maximizing the margin but also minimizing the error terms. To control the classifier tolerance for error, a cost parameter C is introduced as a multiple of the sum of errors. The higher the value of C , the higher the penalty is of an error and the classifier will sacrifice margin for error reduction. The lower the value of C , the classifier will attempt to maximize margin with the possibility of classification errors.

Weka [18], by default, sets this cost parameter to one. We maintained the default setting in our experiments.

2.1.2 Non-linearly-separable data

If our data is non-linearly-separable in the original input space it is defined in, we increase its dimension. We project it onto a higher dimensional space where it becomes linearly separable. Figure 4 illustrates how this projection could linearly separate data in a different space. Two issues arise from transforming data into a higher dimension. (i) The

curse of dimensionality which is absence of enough data points to completely describe the behavior of data in the new space. (ii) The computational cost increases when calculating dot products in a higher dimensional space. This problem is worsened when the new space goes to infinity. The first problem is solved by having a larger training data sample. The second problem is taken care of by the *kernel trick*. Since the SVM algorithm solely depends on dot-products, the kernel trick transforms the linear classifier into a non-linear one by substituting the dot-product operation in the original space with a kernel function that takes in the weight vector and the feature vector as parameters.

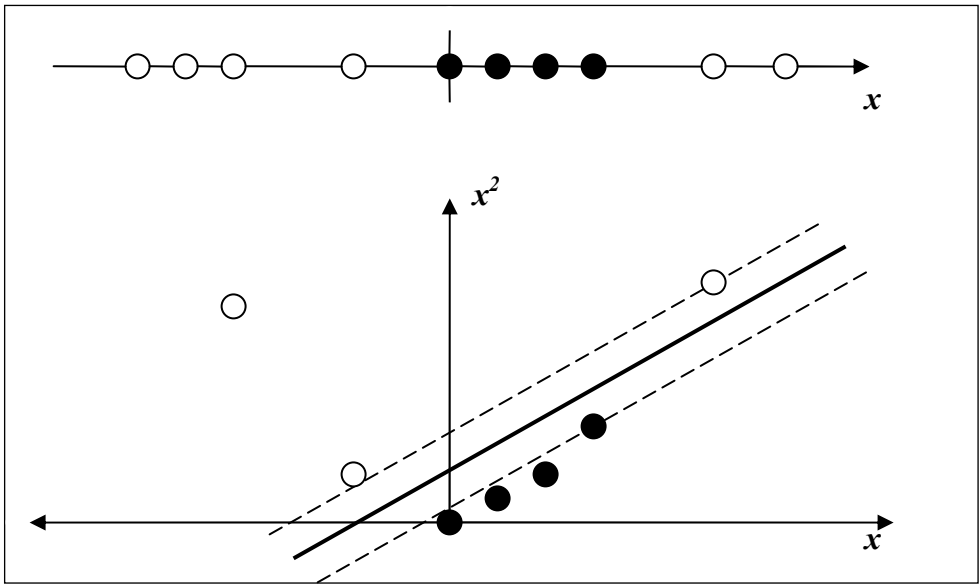


Figure 4 – An increase in dimension leading to linearly separable data

We do not examine non-linear SVM in our research. This is because of computational costs associated with running non-linear SVMs on large data sets.

SVM model	Time taken to build model	Number of kernel evaluations
Linear-SVM	47.9 seconds	6028633
Quadratic SVM	690.48 seconds	227056660
Cubic SVM	981.66 seconds	242518719
RBF (Radial Basis Function) kernel-SVM	5548.67 seconds	1877184328

Table 1 – Computational costs associated with training with linear and non-linear SVMs on data-sample1, a data set of 10,000 instances

We plan to examine non-linear SVMs in future experiments.

2.1.3 Multi-class classification with SVM

Weka's implementation of SVM converts any multi-class classification problem into a binary classification problem by examining instances of two classes at a time and building a model for classification for only these two classes.

2.1.4 Properties of Linear-SVM that we are trying to benefit from

- SVMs are based on statistical learning theories. It simplifies the classification problem to a quadratic programming problem. There are efficient methods for solving QP. Moreover, The QP problem has a unique solution.
- SVMs are guaranteed not to be influenced by local minima as they attempt to find a global maximum-margin hyperplane. This compares with Neural Networks that are prone to get stuck in local minima.
- Linear-SVMs have been used in several applications with performances better than other statistical learning classifiers like Naïve Bayes classifiers [8].
- Linear-SVMs, after training, produce a weight vector that gives us insight into the key features in our data. The higher the absolute weight value for each attribute, the more significant it is on our data set. Non-linear SVMs do not produce such a weight vector; therefore, the model built is difficult to understand by a human expert.
- Linear-SVMs are computationally less expensive than non-linear SVMs, see Table 1.
- Linear-SVMs have a low VC-dimension. If we think of VC-dimension as the tendency of a classifier to exactly represent the training data. The more flexible or

the higher the VC-dimension of the classifier, the more likely it is to *overfit* the training data leading to poor testing results.

2.2 Repeated Incremental Pruning to Produce Error Reduction (RIPPER)

This section provides a summary of RIPPER. A more detailed explanation of how RIPPER works could be found in [2]. RIPPER produces a rule set consisting of a disjunction of conjunctions. This means:

1. Each rule consists of several conditions that are connected by AND operators.
2. Each rule is connected to other rules by OR operators.
3. Each rule, when satisfied, implies a membership in a particular class.

To classify a data instance consisting of different values for each attribute, each rule in the learned rule-set is examined in sequence. The rule for which all conditions are satisfied defines the prediction made for the data instance.

2.2.1 How does RIPPER derive its rule set?

RIPPER is a two-stage algorithm. In the first stage, the algorithm builds a rule-set. In the second stage it optimizes the rule-set. In the building stage, the algorithm partitions the *uncovered* training data set into a *growing* and *pruning* set. An uncovered set consists of all data instances that have not yet been described by a rule's antecedents or conditions. An instance is covered by a rule if it satisfies all the conditions in the rule. Rules are grown from the growing set greedily. A condition is added to a rule (which could be empty) by examining all possible values of an attribute and adding the condition with the highest *information gain*. Intuitively this means, we add the condition that (i) increases the proportion of *positive* examples, or correctly classified and covered

instances over all instances covered by the rule (ii) and does not significantly reduce the number of covered positive examples. A condition takes the following forms (i) Attribute value \geq continuous value (ii) Attribute value \leq continuous value if the attribute is a continuous variable or (iii) Attribute value = nominal value. We stop growing a rule when we can no longer find a rule that adds a positive information gain or the rule only covers positive examples, i.e. it is 100% accurate.

The pruning stage attempts to simplify the rule. It does so by removing a sequence of conditions at the end of the rule. This greedy process examines which deleted sequence maximizes the proportion of positive examples over total examples covered. Note that when we terminate growing a rule when reaching 100% accuracy, we do not prune any tail sequence of conditions from the rule. We add the pruned rule to the rule set.

We now remove all examples covered by the rule-set, re-partition our training set into growing and pruning sets and grow another rule. The decision of when to stop growing more rules is extremely important. An obvious method would be to continue growing rules until we cover all examples in our training set. The problem occurs when our data consists of many noisy data instances leading to a very large rule-set where each rule only covers a small set of positive examples. To solve this problem we stop growing rules when we reach a theoretical bound on the number of rules based on data encoding, known as the Minimum description length¹ (MDL). RIPPER evaluates the description length of the rule set after every rule is added. It examines each rule from the last to the first rule and deletes any rule that increases the rule-set's description length.

¹ The MDL of a rule is the sum of bits required to encode the model and bits required to encode errors made by the model.

The optimization stage examines each rule in sequence and decides whether the rule needs to be replaced, revised or kept. The replacement for a rule is found by growing a rule as before and pruning it on a different measure. We now prune based on whether the deleted sequence reduces the number of errors of the entire rule set, not just the current rule. The revised rule is created by adding more antecedents to the original rule. A decision of which rule to choose is also based on the Minimal description length heuristic. The rule with the smallest MDL is chosen.

2.2.2 Multi-class classification with RIPPER

RIPPER allows for multi-class classification. This is because it is based on a 'set-covering' idea. It attempts to generate rules that cover all positive examples of a class and then moves to the next class until one class is left without rules generated specifically for it. When classification, each rule is examined in sequence, if no rule is satisfied, the data instance is classified as the last class with no rules. It is important to note here, that RIPPER generates rules for the least prevalent set to the most prevalent.

2.2.3 Properties of RIPPER that we are trying to benefit from

- RIPPER is used in industry as a rule based machine learning algorithm
- It is as effective as the rules generated by C4.5 without the large efficiency setbacks [2].
- The rules provided by RIPPER are to a certain extent human readable
- RIPPER makes no statistical assumptions about the data, for example, it does not require linearly separable data guarantee to produce good results.

- It is less sensitive to outliers compared to AdaBoost. This is due to the use of MDL to simplify rules and the search for rules that cover a high proportion and large number of positive examples

Appendix A contains pseudo-code for Weka's implementation of the algorithm.

2.3 Adaptive Boosting with Decision stumps (AdaBoost)

AdaBoost was introduced in 1995 by Freund and Schapire [5]. It is a meta-algorithm, meaning that it integrates base machine learning classifiers. For our research, we examined the performance of AdaBoost with decision stumps, a weak classifier. I'll first explain how decision stumps work and then I'll describe how AdaBoost creates a voted combination of this weak classifier to create a much more complex, better classifier.

2.3.1 Decision stumps

Decision stumps are very simple classifiers. They examine all features or attributes of a training data set and they return a decision tree classifier with two leaves for binary classification and a single root node which evaluates the value of only one feature. A decision stump prediction could be mathematically represented as a function that returns -1 or +1 labels:

$$h(x; w_0, k, w_1) = \text{sign}(w_0 x_k - w_1)$$

'k' represents the single attribute or feature in the feature vector x, that the decision stump pays attention to.

2.3.2 Building a classifier from voted combinations of weak classifiers

AdaBoost runs for a given number T of iterations. At each round, there are two possible outcomes:

- A new decision stump classifier is added with a calculated weight or vote that determines the importance of the added classifier's decision.

- The distribution of the training sample is modified to give higher importance to training instances that are in-correctly classified, forcing the next weak classifier to be added to try to eliminate the training errors.

Initially all training instances have equal weights; therefore, the distribution,

$D_1(i)$, of each instance, i , is equal to $\frac{1}{\text{total number of training instances}}$. The base

classifier's job, on iteration t , is to find a prediction function or *weak hypothesis* h_t that minimizes *weighted-classification error* measured as:

$$e_t = \sum_{i:h_t(x_i) \neq c_i} D_t(i)$$

If the decision stump implementation cannot be given distribution values for each instance, the training set could be re-sampled to create a new sample where the frequency of each instance matches the distribution.

The base classifier returns with a weak hypothesis. First the hypothesis's weighted-classification error is checked. If the hypothesis performs worse than chance that is its error value is greater than half, the AdaBoost program terminates. No more combinations of weak hypothesis could improve the classifier performance. If the hypothesis performs better than chance, it is given a vote based on the number of training

errors it makes. $\alpha_t = \frac{1}{2} \ln \left(\frac{1 - e_t}{e_t} \right)$

The distribution of each instance is now updated based on the new hypothesis's ability to correctly classify each instance. If it misclassifies that data instance, its distribution is increased. If it correctly classifies the data instance, its distribution is

decreased. The rate of increase or decrease is dependent on the importance vote of the new hypothesis.

$$D_{t+1} = \frac{D_t}{Z_t} \times \begin{cases} \exp(\alpha_t) & \text{if hypothesis misclassified instance} \\ \exp(-\alpha_t) & \text{if hypothesis correctly classified instance} \end{cases}$$

Z_t is a normalizing constant to ensure that $\sum D_{t+1}(i) = 1$.

The final hypothesis produced by AdaBoost used for classification returns a weighted majority vote of T weak hypotheses.

2.3.3 Multi-class classification with AdaBoost and Decision stumps

A variant of AdaBoost, known as AdaBoost.M1 is generally used. AdaBoost.M1 performs relatively well when the weak learner is capable of achieving above 50% accuracy even on distributions that heavily weigh hard classification examples [17]. We did not, however, use this variant as decision stumps were incapable of producing good multi-class classifications, this led to zero boosting. Alternatively, we converted our multi-class classification problem into a binary classification problem by classifying one class at a time from all other classes.

2.3.4 Properties of AdaBoost that we are trying to benefit from

- AdaBoost is theoretically proven to decrease training error on each iteration [17].
- Except for the number of iterations, there are no parameters that require fine-tuning with AdaBoost
- Using decision stumps as base classifiers leads to weak hypotheses that are easy to understand and evaluate by a human expert.

- AdaBoost enables us to determine the hardest examples in our data set to classify. These are the examples with the highest distribution values at the termination of AdaBoost.
- AdaBoost combines linear weak-classifiers to create a non-linear classifier in a high dimensional space. AdaBoost simplifies to a linear programming problem which is less computationally expensive to solve compared to the Quadratic Programming problem embedded in Support vector machines.

3 Methodology

3.1 Data collection

In our research we test the performance of the different machine learning classifiers on two different data sources. We term the data sources public and private. The properties of the data sources are as follows:

A *public* data set is a publicly available data set. We used several public data sets from NLANR (National Laboratory for Applied Network Research). These data sets consisted of Time Sequenced Header (TSH) files. A TSH file represents each packet with 44 bytes that include a timestamp, interface number, IPv4 header without options, and the first 16 bytes of the transport header. Timestamps in TSH files may be relative to the UNIX epoch or relative to the card reset; therefore the absolute value of a timestamp is not guaranteed to be correct.

Appendix B contains detailed information of the data sets used and Table 2 gives a summary of their properties.

Data Set Category	COS-CodeRed	COS-Doom	AMP-path
Start date of data collection	Tuesday, July 10 2001	Monday, January 26 2004	Friday, March 11 2005
Min number of packets	301070	881155	759101
Max Number of packets	978182	2243244	8706957
Average number of packets	582601	1562400	4461635
Min proportion of ssh	0.45%	0.02%	1.08%
Max proportion of ssh	1.51%	1.87%	25.27%
Average proportion of ssh	0.91%	0.56%	6.82%

Table 2 – Properties of public data sets used

A *private* data set consists of packets collected internally within our research’s lab network. Our data collection approach does not collect data from monitors placed on the lab’s network router. Instead, we simulate possible network scenarios using one or more computers and collect traffic. For example, we simulate an ssh connection, by setting one computer as an ssh server, and the other computer as an ssh client. We connect using different ssh client applications, run different ssh services such as email tunneling or file transfer and, collect all packets transmitted using ethereal [3]. Figure 5 illustrates the data collection method. For certain applications, where configuring the servers required a high expertise level, we connected to servers on the internet and collected packets on the client computer using ethereal. X-Forwarding was such an application as configuring an X-server was difficult. “SSH, The Secure Shell” [1] was used as a resource to help us configure the ssh servers and clients and simulate different ssh services. We also simulated the following application behaviors: http, https, ftp, chat applications (yahoo, msn), ssh (including scp, sftp, tunneling, X-Forwarding), rdp or remote desktop, streaming media, peer-to-peer (including Kazaa, Gnutella), telnet, smtp, pop3 and imap. All generated files are stored in .pcap format. Pcap files include all information about a

packet including all headers, application payload, as well as a timestamp value indicating packet-arrival time on the network card. Table 3 gives relative percentages of different applications in the generated data sets.

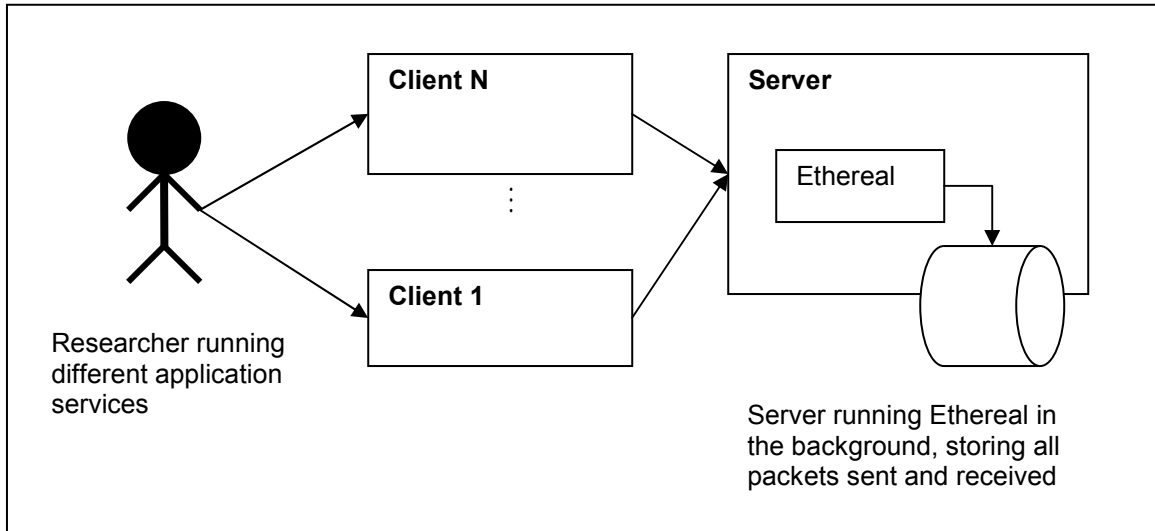


Figure 5 – Simulating network application on local network and collecting data

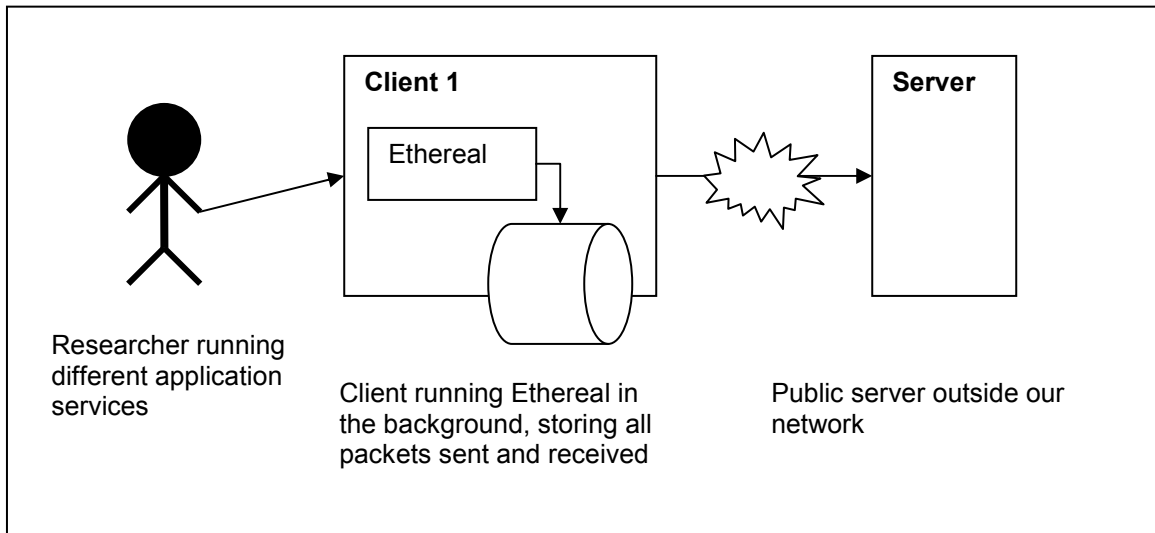


Figure 6 – Using servers external to local network for simulating application behavior

DataSet	Total size (# packets)	ssh	http	p2p	mail	ftp	rdp	telnet	chat	real
data-10000-sample1.arff	10000	0.25	0.25	0.3	0.05	0.05	0.03	0.023	0.023	0.024
data-10000-	10000	0.2	0.25	0.3	0.05	0.1	0.03	0.023	0.023	0.024

sample2.arff											
data-10000-sample3.arff	10000	0.15	0.25	0.3	0.1	0.1	0.03	0.023	0.023	0.024	
data-10000-sample4.arff	10000	0.1	0.25	0.3	0.1	0.1	0.05	0.04	0.03	0.03	
data-10000-sample5.arff	10000	0.05	0.25	0.3	0.15	0.15	0.05	0.016	0.017	0.017	

Table 3 – Proportions of each network application in private data sets

3.2 Feature selection

It is important to look at the structure of IP, TCP and UDP headers before discussing the features selected for testing with the different classifiers. Figure 7 - Figure 9 include all header information for reference. These figures have been created from the header definitions in [14, 15, 16].

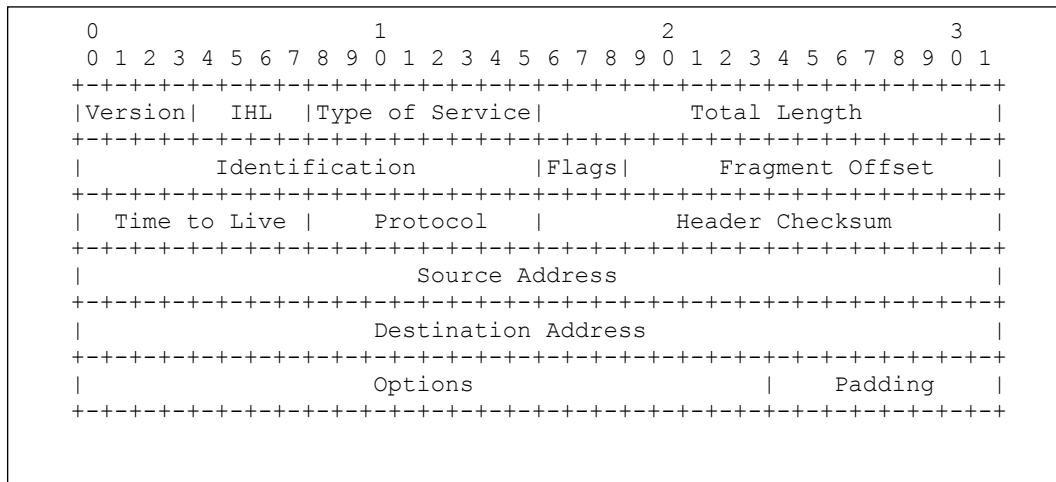


Figure 7 – IP header

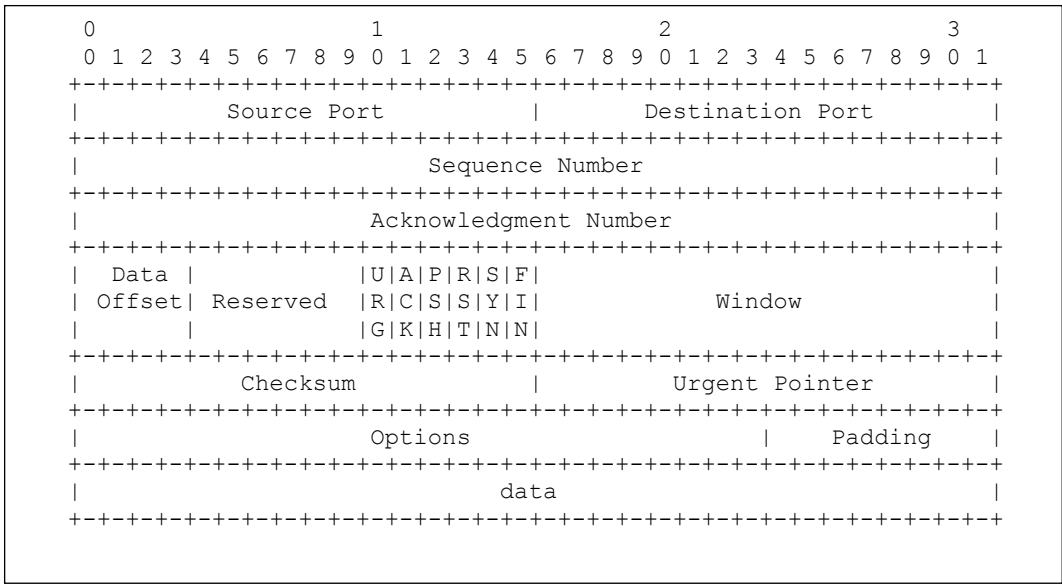


Figure 8 – TCP header

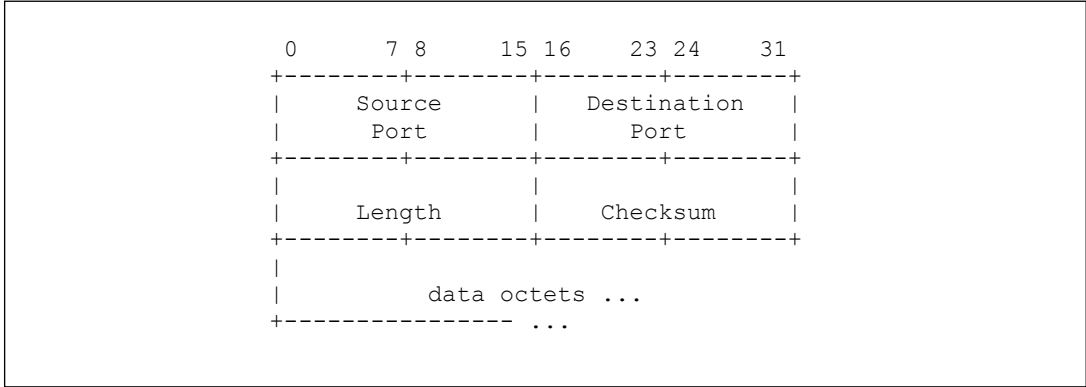


Figure 9 – UDP header

Public data sets do not contain payload due to privacy issues. Our private data sets, however, do contain payload information along with headers. To select features from the payload, we could deal with payload as if it is a regular document and apply feature selection algorithms that examine terms present in all collected payload data. There are several feature selection algorithms that have been applied in text categorization problems such as Mutual Information and Chi-Square. In our experiments we did not examine payload data. Payload data could provide us with information that could significantly boost the classifiers’ performances. Due to time constraints, examining payload is left as an extension to the current research.

Our strategy for selecting features from the headers is to avoid extensive fine tuning of features. Here are our reasons:

- Different feature selection algorithms could affect the performance of different machine learning algorithms in different ways. We would like to limit the variables effecting performance in our comparative study.
- Our data sets contain several numeric values, what is the best way to discretize the data for feature selection algorithms like Mutual Information to work appropriately? [11] propose methods for the Chi-Square feature selection algorithm that automatically discretize numeric features. However a number of features deal with sequence such TCP sequence number, TCP Acknowledgment number and IP Time to live. Would discretization lose a lot of relevant information?

There are relatively few features. The *curse of dimensionality* is not an issue as we have extremely large public training sets with around 300,000 data instances.

- Appendix B contains exact details.

Looking at feature selection algorithms is definitely important. However, this is left as future work to this investigatory research study.

From headers, the following 13 attributes were used as features:

1. Inter-arrival time – calculation details are provided in the Data preparation section.
2. IP header length – IHL, bits [4, 7] in the IP header, the minimum header length is 5. If the length is greater than 5, it indicates the presence of options in the IP header. Most applications do not use IP options. SSH, definitely does not use IP options.
3. IP Fragment flags – Flags, bits [48, 50] in the IP header. The first bit must always be zero, therefore the possibilities are 0 – indicating last fragment and packet could be fragmented, 1 – indicating more fragments on the way, 2 – indicating last fragment and do not fragment packet, 3 – indicating do not fragment packets and that more fragments are on the way.
4. IP Time To Live – TTL, bits [64, 71] is a value set by the sender of the packet, which is reduced on every point along the route, when time to live reaches 0, the packet is destroyed.
5. IP Protocol – Protocol, bits [72, 79] indicates the upper layer protocol that should process the packet. ‘6’ indicates TCP and ‘17’ indicates UDP. We remove any other protocol from the datasets.

6. TCP Source Port – Source Port, bits [0, 15] in the TCP header and bits [0, 15] in the UDP header. Indicates the source port number. Note that from an ssh server to the client, this field would be 22 if the server is running on the ssh well-known port number.
7. TCP Destination Port – Destination port, bits [16, 31] in the TCP header and bits [16, 31] in the UDP header. Indicates the destination port number. Note that from an ssh client to an ssh server, this field would be set to 22 if the server listens on the ssh well-known port number.
8. TCP Sequence number – The sequence number of the first 8 bits in the TCP segment. Gives a good idea of the position of the packet within the connection. This is blank for UDP packets. Note that UDP is a connection-less protocol and therefore, sequence number is irrelevant.
9. TCP Acknowledgment number – The next expected sequence number the sender is expecting to receive, also gives a good idea of the position of the packet within the connection. This is blank for UDP packets. Note that UDP is a connection-less protocol and therefore, sequence number is irrelevant.
10. TCP Header Length – data-offset, bits [96, 99] in the TCP header and set to 8 for UDP header. This indicates the presence of options in the TCP header; otherwise the minimum header length is 20 for TCP.
11. TCP control bits – TCP only, bits [100, 111] in the TCP header. The 6 bits are flags used to indicate:
 - a. Urgent pointer field significant

- b. Acknowledgment field significant
- c. Push function
- d. Reset the connection
- e. Synchronize sequence numbers
- f. No more data from sender

12. TCP window – TCP only, bits [112, 127] indicate the number of bytes the sender is capable of receiving. Hypothetically, this gives us a good indication of the different roles in a connection. For example a file transfer server might have a large window size indicating a network capable of dealing with more packets compared to the client machine that is only capable of accepting a limited number of packets. However this value is also dependent on the current condition of the network. TCP is a network-friendly protocol if the network is busy, both the client and the server shrink their window size to limit the number of packets being sent at a time.

13. Payload length – The size of the payload. This feature is important as it would give insight on the type of application running. With file transfer services we expect larger payload sizes. Details of calculating this field are provided in the Data preparation section.

In certain experiments TCP Sequence and Acknowledgment numbers as well as Source and destination ports were removed. This is because we found all classifiers to overfit the private data sets and exclusively train on these features. For example, when attempting to find the different services running within ssh traffic, the port number on the

client side was different for each application service and this port number was used to determine the service. Also the different connections' lengths for each service led to a certain range of TCP sequence and acknowledgment numbers for the different services. To ensure that the classifiers were not taking advantage of this bias within the data set, we eliminated these four features.

3.2.1 Why did we not choose all fields in the headers?

Although we decided not to fine tune any of the features available from header fields, certain fields had to be eliminated. Here are the reasons for eliminating these fields:

- Values are the same for all data instances. IP version number was always 4. Type of service was not modified. TCP urgent pointer did not exist in any of the fields.
Reserved bits in the TCP header were not modified.
- Values are irrelevant to classification. IP identification is a random number generated that is irrelevant to the type of application the packet belongs to. Similarly IP, TCP and UDP checksums are used for transmission security and do not depend on the type of application.
- Values not provided. Options were stripped from TSH files provided by public data sets.
- Values are obfuscated or biased. Exact Source and Destination IP address are hidden in public data sets, only the network part of the address is revealed. In the private data sets, the machines used had fixed IP addresses and were within the same network leading to an artificial, possibly biased data set.

3.3 Data preparation

Weka provides an easy to use interface for the three different machine learning algorithms we are examining. It expects an Attribute-Relation file format (ARFF) data format. Figure 10 is a sample ARFF file.

```
@relation ssh-weka.filters.unsupervised.instance.Resample-S1-Z100.0

@attribute interarrivaltime numeric
@attribute ipheaderLength numeric
@attribute ipFragmentFlags {0,1,2,3}
@attribute ipTTL numeric
@attribute ipProtocol numeric
@attribute tcpSourcePort numeric
@attribute tcpDestinationPort numeric
@attribute tcpSyn numeric
@attribute tcpAck numeric
@attribute tcpHeaderLength numeric
@attribute tcpControlBits numeric
@attribute tcpWindow numeric
@attribute payloadLength numeric
@attribute isSSH {true,false}

@data

0.00008,5,0,127,6,1866,5007,2282500141,1419416363,5,16,17520,0,false
```

Figure 10 – Snapshot from a sample ARFF file used in our experiments

The first step of data preparation is converting a Time Sequenced Header file or a PCAP file into an ARFF file. This is a simple process that involves writing a comma-separated string of features followed by the class label. The following sections describe how certain fields that are not easy to derive directly from the headers are calculated. The data conversion code is written in Java and is available on request. A java packet capture program was used to read PCAP files [6].

3.3.1 Calculating inter-arrival time

Both the TSH and PCAP files provide absolute timestamp values. Absolute timestamp is an unreliable measure as: (i) it is dependent on the time set on the network card which is not synchronized with a central time. (iii) The time on the card could be

easily reset. (ii) Time of day could provide important information on the type of application typically running; however, in our private data sets, applications were not run during typical hours of the day.

We decided to use a different metric: inter-arrival time measures the difference in milliseconds between the previous packet and the current packet sent from the same host within the same session. Two packets are within the same session, if the ip destination and source addresses and destination and source ports match. We sort TSH and PCAP files by timestamp intermediately to ensure that the inter-arrival times are non-negative.

3.3.2 Calculating payload length

There is no direct field indicating payload size in the headers. Payload length is calculated as follows:

$$Payload\ length = \begin{cases} IP\ Total\ Length - (IPIHL \times 4) - (TCP\ Header\ Length \times 4) & \text{if TCP packet} \\ IP\ Total\ Length - 8 & \text{if UDP packet} \end{cases}$$

3.3.3 Class labeling for training and test validation

With public data sets, there is no accurate method of determining the type of application the different packets belong to. We use a port-based classifier to heuristically determine the class label. The pseudo-code for the port-based classifier is as follows:

If (destination port = 22 \vee source port = 22) then packet is SSH

Else not SSH

Classifying private data sets could be done accurately because we know exactly the applications running in every simulation. With the help of filters in Ethereal, we ensured that any noise traffic, such as ARP packets, is removed from the pcap file. We label non-ssh packets as false. Depending on the purpose of the data set generated, we

label ssh packets as true or we label the different services simulated. There are 5 different service labels

- scp – indicating secure copy simulation
- sftp – indicating secure file transfer simulation
- xForward – indicating X-Forwarding session
- tunnel – indicating local, remote or dynamic port forwarding of different application within ssh
- ssh – indicating a normal ssh remote login session

3.3.4 Final data preparation steps

To remove any ordering that could affect performance; we use Weka's unsupervised random re-sampling command on any data set before training or testing.

4 Experiments

4.1 Phase 1: Differentiating ssh from other applications

4.1.1 Experiment 1: Comparing performance of Linear-SVM, AdaBoost with Decision stumps, RIPPER and J48 on public data sets

Our goal here is to find the best performing machine learning algorithm given the following 13 features:

1. Inter-arrival time
2. IP header length
3. IP Fragment flags
4. IP Time To Live
5. IP Protocol
6. TCP Source Port (or UDP source port)
7. TCP Destination Port
8. TCP Sequence number
9. TCP Acknowledgment number
10. TCP Header Length
11. TCP control bits
12. TCP window
13. Payload length

We evaluate the performance of each classifier based on two measures:

Recall: The number of correctly classified ssh packets from all ssh packets

Precision: The number of correct ssh packets from all packets classified as ssh

A better classifier will have higher recall and precision values. We trained each classifier on the same data set and tested them on data sets belonging to the same network. Our choice of training data sets was based on computational costs. Data sets containing about four million or more instances produced Out-of-memory exceptions on our server which has around 32GB of RAM.

4.1.1.1 Results

Train set	Test set	Precision				Recall			
		<i>LSVM</i>	<i>AdaBoost</i>	<i>RIPPER</i>	<i>J48</i>	<i>LSVM</i>	<i>AdaBoost</i>	<i>RIPPER</i>	<i>J48</i>
COS-994758814	COS-994725605	0.136	0.911	1	1	0.457	0.818	1	1
	COS-994736662	0.325	0.848	1	1	0.548	0.927	1	1
	COS-994745933	0.602	0.986	1	1	0.576	0.894	1	1
	COS-994758814	0.905	0.989	1	1	0.677	0.955	1	1
	COS-994769861	0.5	0.993	1	1	0.742	0.997	1	1
	COS-994777339	0.176	0.951	1	1	0.533	0.893	1	1
	COS-994790224	0.303	0.273	1	1	0.293	0.729	1	1
	COS-994801283	0.272	0.978	1	1	0.577	0.968	1	1
COS-994769861	COS-994725605	0.294	0.821	1	0.806	0.365	0.817	1	0.89
	COS-994736662	0.605	1	1	0.337	0.38	0.867	1	0.936
	COS-994745933	0.817	1	0.998	0.993	0.345	0.777	1	0.924
	COS-994758814	0.955	1	0.989	0.995	0.404	0.908	1	0.923
	COS-994769861	0.736	0.987	1	1	0.445	0.994	1	0.999
	COS-994777339	0	0.998	0.999	0.497	0	0.891	1	0.894
	COS-994790224	0.74	0.997	1	0.983	0.218	0.497	1	0.752
	COS-994801283	0.632	1	1	0.2	0.426	0.963	1	0.969
COS-1075086697-	COS-1075086697-1	0.726	0.995	1	1	0.787	0.989	1	1
	COS-1075099560-1	0.025	0.593	0.999	1	0.995	1	1	1
	COS-1075110637-1	0.692	0.989	1	1	0.917	0.917	1	1
	COS-1075119912-1	0.028	0.862	0.971	1	0.064	0.999	1	1
	COS-1075142054-1	0.126	0.034	0.922	1	0.999	0.573	1	1
	COS-1075151298-1	0.02	0.845	0.996	1	0.055	0.482	1	1
COS-1075099560-	COS-1075086697-1	0.868	0.97	1	1	0.786	0.898	1	1
	COS-1075099560-1	0.855	0.99	1	1	0.916	0.916	1	1
	COS-1075110637-1	0.042	0.569	1	1	0.908	0.908	1	1
	COS-1075119912-1	0.222	0.027	0.971	1	0.87	0.391	1	1
	COS-1075142054-1	0.045	0.844	0.921	1	0.061	0.87	1	1
	COS-1075151298-1	0.033	0.484	0.996	1	0.049	0.371	1	1
AMP-1110523221-1	AMP-1110523221-1	0.958	0.992	1	1	0.683	0.932	1	1
	AMP-1110532626-1	0.004	0.969	1	1	0	0.666	1	1
	AMP-1110545630-1	0.002	0.971	1	1	0	0.026	1	1
	AMP-1110555035-1	0.015	0.992	1	1	0	0.394	1	1
	AMP-1111008395-1	0.009	0.933	1	1	0	0.331	1	1
	AMP-1110564441-1	0.01	0.188	1	1	0	0.037	1	1
	AMP-1110967176-1	0.002	0.362	1	1	0	0.059	1	1
	AMP-1110577446-1	0.005	0.724	1	1	0	0.514	1	1
AMP-111053262	AMP-1110523221-1	0.37	0.992	1	1	0.054	0.932	1	1
	AMP-1110532626-1	0.771	0.969	1	1	0.694	0.666	1	1
	AMP-1110545630-1	0.106	0.933	1	1	0.007	0.331	1	1
	AMP-1110555035-1	0.007	0.971	1	1	0	0.026	1	1
	AMP-1111008395-1	0.528	0.992	1	1	0.007	0.394	1	1

	AMP-1110564441-1	0.468	0.188	1	1	0.033	0.037	1	1
	AMP-1110967176-1	0.203	0.724	1	1	0.043	0.514	1	1
	AMP-1110577446-1	0.444	0.362	1	1	0.015	0.059	1	1

Table 4 – Precision and recall values of the different classifiers when trained and tested on different public data sets

	LSVM	AdaBoost	RIPPER	J48
LSVM	<i>t – stat</i>	-6.68552	-12.8132	-10.607
	<i>P(T<=t) one-tail</i>	1.14E-09	1.41E-16	4.91E-16
	<i>t Critical one-tail</i>	1.662979	1.681071	1.669014
AdaBoost		<i>t – stat</i>	-4.3962	-2.93652
		<i>P(T<=t) one-tail</i>	3.55E-05	0.002254
		<i>t Critical one-tail</i>	1.681071	1.667238
RIPPER			<i>t – stat</i>	1.708387
			<i>P(T<=t) one-tail</i>	0.047306
			<i>t Critical one-tail</i>	1.68023
J48				

Table 5 – Student’s t-tests assuming unequal variances of precision values of classifiers on public data sets

	LSVM	AdaBoost	RIPPER	J48
LSVM	<i>t – stat</i>	-4.15986	-12.1378	-11.7289
	<i>P(T<=t) one-tail</i>	3.76E-05	8.81E-16	1.95E-15
	<i>t Critical one-tail</i>	1.662765	1.681071	1.68023
AdaBoost		<i>t – stat</i>	-6.84766	-6.45663
		<i>P(T<=t) one-tail</i>	1.07E-08	3.27E-08
		<i>t Critical one-tail</i>	1.681071	1.679427
RIPPER			<i>t – stat</i>	0.011748
			<i>P(T<=t) one-tail</i>	1.681071
			<i>t Critical one-tail</i>	0.023496
J48				

Table 6 – Student’s t-tests assuming unequal variances of recall values of classifiers on public data sets

4.1.1.2 Discussion

The t-test tables provide three values, the calculated t-value and the probability that our t-value is less than the critical t-value at a calculated degree of freedom and level of significance, $p = 0.05$ and the critical t-value. We only provide the one-tailed t-test as we are interested in testing whether the recall and precision of one classifier is worse (or

better) than the other. From Table 5 and Table 6, we realize a statistically significant difference in the recall and precision means between Linear-SVMs and the other classifiers. Linear-SVMs are worse than the other three classifiers (indicated by the negative t-test values). RIPPER and J48 perform better than AdaBoost as the calculated t-values exceed the critical one-tail t-value. There is no statistically significant difference between RIPPER and J48.

4.1.2 Experiment 2: Comparing performance of Linear-SVM, AdaBoost with Decision stumps, RIPPER and J48 on private data sets

As in the previous experiment, we also examine the performance of the different classifiers. With our private data sets, however, the class-labels are accurate and not based on the heuristic used to determine class labels for public data sets, see section ‘Class labeling for training and test validation’. We also examine other classifier features in detail with our private data sets, such as the effect of varying the percentage of ssh within each training set on the classifier’s performance.

The properties of the private data sets are described in Table 3. The features used are as in the previous experiment; all 13 features.

4.1.2.1 Results

Train set	Test set	Precision				Recall			
		<i>LSVM</i>	<i>AdaBoost</i>	<i>RIPPER</i>	<i>J48</i>	<i>LSVM</i>	<i>AdaBoost</i>	<i>RIPPER</i>	<i>J48</i>
data-sample1	data-sample1	0.758	0.994	1	1	0.725	0.997	1	0.999
data-sample1	data-sample2	0.743	0.989	0.997	0.997	0.705	0.996	0.996	0.996
data-sample1	data-sample3	0.689	0.982	0.999	0.998	0.727	0.995	0.996	0.997
data-sample1	data-sample4	0.569	0.986	0.998	0.995	0.746	0.999	0.999	0.999

data-sample1	data-sample5	0.405	0.955	1	0.992	0.723	1	0.998	1
data-sample2	data-sample1	0.729	0.994	0.998	0.996	0.714	0.997	0.997	0.998
data-sample2	data-sample2	0.739	0.991	1	0.999	0.698	0.996	1	1
data-sample2	data-sample3	0.679	0.982	0.998	0.994	0.721	0.995	0.997	0.997
data-sample2	data-sample4	0.553	0.986	0.997	0.992	0.744	0.999	1	1
data-sample2	data-sample5	0.394	0.955	0.994	0.982	0.723	1	0.998	1
data-sample3	data-sample1	0.745	0.994	0.997	0.998	0.697	0.997	0.997	0.997
data-sample3	data-sample2	0.761	0.991	0.994	0.998	0.686	0.996	0.996	0.996
data-sample3	data-sample3	0.695	0.982	0.991	0.999	0.696	0.995	0.997	0.995
data-sample3	data-sample4	0.571	0.986	0.991	0.999	0.723	0.999	0.999	0.999
data-sample3	data-sample5	0.398	0.957	0.982	0.988	0.693	1	1	1
data-sample4	data-sample1	0	0.994	0.997	0.995	0	0.997	0.997	0.997
data-sample4	data-sample2	0	0.991	0.995	0.994	0	0.996	0.996	0.996
data-sample4	data-sample3	0	0.982	0.991	0.992	0	0.995	0.995	0.995
data-sample4	data-sample4	0	0.986	0.995	0.997	0	0.999	0.999	0.999
data-sample4	data-sample5	0	0.957	0.982	0.968	0	1	1	1
data-sample5	data-sample1	0	0.996	1	0.999	0	0.997	0.997	0.997
data-sample5	data-sample2	0	0.996	0.997	0.997	0	0.996	0.996	0.996
data-sample5	data-sample3	0	0.991	0.999	0.998	0	0.995	0.994	0.995
data-sample5	data-sample4	0	0.995	0.998	0.997	0	0.999	0.999	0.999
data-sample5	data-sample5	0	0.984	1	0.996	0	1	1	1

Table 7 – Precision and Recall (of classifying ssh) values of classifiers on private data sets

	LSVM	AdaBoost	RIPPER	J48
LSVM	<i>t – stat</i>	-9.14821	-9.33191	-9.31289
	<i>P(T<=t) one-tail</i>	1.36E-09	9.32E-10	9.69E-10
	<i>t Critical one-tail</i>	1.710882316	1.710882316	1.710882316
AdaBoost	<i>t – stat</i>		-4.16067	-3.54925
	<i>P(T<=t) one-tail</i>		0.000117	0.000548
	<i>t Critical one-tail</i>		1.695518677	1.688297289
RIPPER			<i>t – stat</i>	0.712487
			<i>P(T<=t) one-tail</i>	0.239962
			<i>t Critical one-tail</i>	1.68023
J48				

Table 8 – Student’s t test, assuming unequal variance for precision values of classifiers on private data sets

	LSVM	AdaBoost	RIPPER	J48
LSVM	<i>t – stat</i>	-7.948687	-7.95317	-7.9554
	<i>P(T<=t) one-tail</i>	1.77E-08	1.75E-08	1.74E-08
	<i>t Critical one-tail</i>	1.710882316	1.710882316	1.710882316
AdaBoost	<i>t – stat</i>		-0.61387	-0.91081

		<i>P(T<=t) one-tail</i>	0.271099	0.183474
		<i>t Critical one-tail</i>	1.677224191	1.677224191
RIPPER			<i>t – stat</i>	-0.31211
			<i>P(T<=t) one-tail</i>	0.378154
			<i>t Critical one-tail</i>	1.677224
J48				

Table 9 – Student’s t test assuming unequal variance for recall values of classifiers on private data sets

4.1.2.2 Discussion

The t-test tables provide three values, the calculated t-value and the probability that our t-value is less than the critical t-value at a calculated degree of freedom and level of significance, $p = 0.05$ and the critical t-value. We only provide the one-tailed t-test as we are interested in testing whether the recall and precision of one classifier is worse (or better) than the other. From Table 8 and Table 9, we realize a statistically significant difference in the recall and precision means between Linear-SVMs and the other classifiers. Linear-SVMs are worse than the other three classifiers (indicated by the negative t-test values). RIPPER and J48 perform better than AdaBoost in terms of precision as the calculated t-values exceed the critical one-tail t-value. However, there is no statistically significant difference between AdaBoost, RIPPER and J48 in terms of recall.

LSVM performed poorly on both public and private data sets. On investigating the reason behind this, we arrive at an important property of our network traffic data sets: using the features we selected our data is non-linearly separable. If we examine port numbers we could understand why this is the case. A packet with either destination or source port 22 is usually ssh. Port numbers less or greater than 22 are not ssh, this leads to a scenario similar to the one in Figure 4. An increase in dimension, or the use of non-

linear SVM could improve performance significantly. Thus, we plan on further investigating non-linear SVMs to solve this problem.

4.1.3 Experiment 3: Effect of reducing proportion of ssh in private data sets on the performance of SVM

Network traffic data generally contains very small proportions of ssh traffic. According to the public data sets, this proportion is around 1%. With small data sets, the sensitivity of the classifier to the proportion of the application becomes an issue. We present the results of our experiment to determine which classifiers are most sensitive to the proportion of the application.

4.1.3.1 Results

		Precision				
	Test set \ Train set	<i>data-sample1</i> (25%)	<i>data-sample2</i> (20%)	<i>data-sample3</i> (15%)	<i>data-sample4</i> (10%)	<i>data-sample5</i> (5%)
	LSVM	data-sample1 (25%)	0.758	0.729	0.745	0
data-sample2 (20%)		0.743	0.739	0.761	0	0
data-sample3 (15%)		0.689	0.679	0.695	0	0
data-sample4 (10%)		0.569	0.553	0.571	0	0
data-sample5 (5%)		0.405	0.394	0.398	0	0
Two tail - critical value		1.859548	1.859548	1.859548		
t value		0.150957	0.161612	9.350479		
P(t <= T) two tail		0.8837471	0.8756183	1.392E-05		
P(t <= T) one tail		0.4418736	0.4378091	6.96E-06		
AdaBoost	data-sample1 (25%)	0.994	0.994	0.994	0.994	0.996
	data-sample2 (20%)	0.989	0.991	0.991	0.991	0.996
	data-sample3 (15%)	0.982	0.982	0.982	0.982	0.991
	data-sample4 (10%)	0.986	0.986	0.986	0.986	0.995
	data-sample5 (5%)	0.955	0.955	0.957	0.957	0.984
	Two tail - critical value	1.859548	1.859548	1.859548	1.859548	
	t value	0.040996	0.041757	2.84E-07	1.492431	
	P(t <= T) two tail	0.9683038	0.9677155	1	0.1739322	

	P(t ≤ T) one tail	0.4841519	0.4838577	0.5	0.0869661	
RIPPER	data-sample1 (25%)	1	0.998	0.997	0.997	1
	data-sample2 (20%)	0.997	1	0.994	0.995	0.997
	data-sample3 (15%)	0.999	0.998	0.991	0.991	0.999
	data-sample4 (10%)	0.998	0.997	0.991	0.995	0.998
	data-sample5 (5%)	1	0.994	0.982	0.982	1
	Two tail - critical value	1.859548	1.859548	1.859548	1.859548	
	t value	1.227882	2.375264	0.272165	2.476418	
	P(t ≤ T) two tail	0.2543908	0.0448763	0.7923872	0.0383253	
	P(t ≤ T) one tail	0.1271954	0.0224381	0.3961936	0.0191627	
J48	data-sample1 (25%)	1	0.996	0.998	0.995	0.999
	data-sample2 (20%)	0.997	0.999	0.998	0.994	0.997
	data-sample3 (15%)	0.998	0.994	0.999	0.992	0.998
	data-sample4 (10%)	0.995	0.992	0.999	0.997	0.997
	data-sample5 (5%)	0.992	0.982	0.988	0.968	0.996
	Two tail - critical value	1.859548	1.859548	1.859548	1.859548	
	t value	1.188662	1.061303	1.249578	1.522701	
	P(t ≤ T) two tail	0.268658	0.3195382	0.2467721	0.1663329	
	P(t ≤ T) one tail	0.134329	0.1597691	0.123386	0.0831664	

Table 10 – Precision values of different classifiers as the proportion of ssh decreases, t-tests of whether effect of proportion reduction on classifiers is statistically significant or not

Table 10 shows the effect of reducing ssh proportions in the private data set on the performance of the different classifiers. While Linear-SVMs are very sensitive, with every reduction in proportion causing a statistically significant change in classifier performance, the other classifiers are stable and show no statistically significant change in performance with the change of ssh proportions in the training data sets.

		Recall					
LSVM	Test set \ Train set	<i>data-sample1 (25%)</i>	<i>data-sample2 (20%)</i>	<i>data-sample3 (15%)</i>	<i>data-sample4 (10%)</i>	<i>data-sample5 (5%)</i>	
		data-sample1 (25%)	0.725	0.714	0.697	0	0
		data-sample2 (20%)	0.705	0.698	0.686	0	0

	data-sample3 (15%)	0.727	0.721	0.696	0	0
	data-sample4 (10%)	0.746	0.744	0.723	0	0
	data-sample5 (5%)	0.723	0.723	0.693	0	0
	Two tail - critical value	1.859548	1.859548	1.859548		
	t value	0.525978	2.154557	5000000		
	P(t <= T) two tail	0.6131685	0.0633244	4.87E-14		
	P(t <= T) one tail	0.3065842	0.0316622	2.435E-14		
AdaBoost	data-sample1 (25%)	0.997	0.997	0.997	0.997	0.997
	data-sample2 (20%)	0.996	0.996	0.996	0.996	0.996
	data-sample3 (15%)	0.995	0.995	0.995	0.995	0.995
	data-sample4 (10%)	0.999	0.999	0.999	0.999	0.999
	data-sample5 (5%)	1	1	1	1	1
	Two tail - critical value	1.859548	1.859548	1.859548	1.859548	
	t value	2.84E-07	2.84E-07	2.84E-07	2.84E-07	
	P(t <= T) two tail	1	1	1	1	
P(t <= T) one tail	0.5	0.5	0.5	0.5		
RIPPER	data-sample1 (25%)	1	0.997	0.997	0.997	0.997
	data-sample2 (20%)	0.996	1	0.996	0.996	0.996
	data-sample3 (15%)	0.996	0.997	0.997	0.995	0.994
	data-sample4 (10%)	0.999	1	0.999	0.999	0.999
	data-sample5 (5%)	0.998	0.998	1	1	1
	Two tail - critical value	1.859548	1.859548	1.859548	1.859548	
	t value	0.572078	0.6	0.338061	0.141421	
	P(t <= T) two tail	0.5829831	0.5651101	0.7440172	0.8910335	
P(t <= T) one tail	0.2914916	0.282555	0.3720086	0.4455167		
J48	data-sample1 (25%)	0.999	0.998	0.997	0.997	0.997
	data-sample2 (20%)	0.996	1	0.996	0.996	0.996
	data-sample3 (15%)	0.997	0.997	0.995	0.995	0.995
	data-sample4 (10%)	0.999	1	0.999	0.999	0.999
	data-sample5 (5%)	1	1	1	1	
	Two tail - critical value	1.859548	1.859548	1.859548	1.859548	
	t value	0.825137	1.425394	2.84E-07	0.500452	
	P(t <= T) two tail	0.4332074	0.1918703	1	0.6302315	
P(t <= T) one tail	0.2166037	0.0959352	0.5	0.3151158		

Table 11 - Recall values of different classifiers as the proportion of ssh decreases, t-tests of whether effect of proportion reduction on classifiers is statistically significant or not

Table 11 shows that reducing the proportion of ssh in training data effects LSVM recall values, just as it effects precision values.

4.1.3.2 Discussion

The sensitivity of the classifier to the proportion of applications within the training data is a key factor when dealing with network traffic data. Applications like ssh occupy a very small percentage of network traffic. If the classifier cannot deal with varying or small proportions of classes within the training data set, it is impractical for general network use. As the network characteristics change, a Linear SVM classifier built for a particular network may perform very poorly if the data sets it trains on, do not contain appropriate proportions of each application.

4.1.4 Experiment 4: Effect of eliminating port numbers on the performance of RIPPER

RIPPER learns rules that are equivalent to the port based classifier we used as a heuristic to label public data sets. That is, it learns that ssh runs on port 22, therefore when either the destination port or the source port is 22, RIPPER achieves 100% accuracy with these two rules on public data sets. We wanted to examine its performance without port numbers. Would RIPPER be capable of deriving rules relevant to ssh classification without using port numbers and still produce high accuracy levels, measured by high recall and precision values? The following results show that on our private data set, RIPPER produces promising results.

4.1.4.1 Results

Train set	Test set	% Correct	Precision	Recall	F-measure
data-sample1	data-sample1	99.68	0.989	0.998	0.993
	data-sample2	98.71	0.966	0.968	0.967
	data-sample3	98.93	0.953	0.974	0.963

	data-sample4	99.11	0.946	0.966	0.955
	data-sample5	99.18	0.875	0.971	0.92

Table 12 – Percentage of correctly identified instances, Precision, Recall and F-measure values for detecting ssh with RIPPER in data sets that do not have port numbers

The following list represents the two most covering rules RIPPER derived to classify ssh:

1. (tcpControlBits >= 24) and
 - o (tcpWindow <= 24820) and
 - o (payloadLength <= 116) and
 - o (payloadLength >= 48) and
 - o (payloadLength <= 52) and
 - o (tcpSyn <= 3061706473) and
 - o (tcpAck <= 3061706733)
 - (787.0/1.0)
2. (tcpWindow <= 24820) and
 - o (tcpWindow >= 24820)
 - (383.0/0.0)
 -
3. (tcpWindow <= 17489) and
 - o (ipTTL >= 127) and
 - o (ipTTL <= 127) and
 - o (tcpAck >= 459874763) and
 - o (tcpWindow >= 16396)
 - (285.0/4.0)

4.1.4.2 Discussion

While the rules learned result in high precision and recall rates, the major problem with the rules is that they are dependent on the network the training sets were derived from making it hard to use the classifier on different networks. For example, tcp window values are based on the condition of the network, with network overload, this value is reduced significantly. Also, IP Time to live gives insight into the position of computers within the network by given hop count details. Therefore rules 2 and 3 cannot be used on packets from different networks. Running RIPPER on a public data set (COS-994758814-1) led to the following rules, which are also network dependent. A ten-fold cross validation test gave a precision value of 0.999, a recall value of 0.995 and an F-measure of 0.997. The percentage of correctly classified instances was 99.9917 %.

1. (tcpWindow >= 33580) and
 - a. (tcpHeaderLength >= 5) and
 - b. (tcpWindow <= 62780) and
 - c. (ipTTL >= 56) and
 - d. (ipTTL <= 56)
 - i. (1857.0/1.0)

2. (tcpWindow >= 62780) and
 - e. (tcpWindow <= 62780) and
 - f. (ipTTL >= 61)
 - i. (1825.0/0.0)
3. (ipTTL >= 250) and
 - g. (ipTTL <= 251) and
 - h. (tcpSyn >= 2784032103)
 - i. (337.0/0.0)

4.2 Phase 2: Differentiating different ssh services

4.2.1 Experiment 1: Comparing classifier's performance for differentiating ssh services

With this experiment, we are limited to using our private data sets as we could accurately determine the type of service running each packet belongs to. The distribution of each service type in the three data samples are shown in table x. Each data set is different from the other, in that each set contains a different time phase of the simulated connections. The first data set, for example, contains the first 2000 packets of a secure file transfer simulation.

For this experiment we limited the number of features we are using from thirteen to the following nine features:

1. Inter-arrival time
2. IP header length
3. IP Fragment flags
4. IP Time To Live
5. IP Protocol
6. TCP Header Length
7. TCP control bits
8. TCP window
9. Payload length

We eliminated destination and source ports. One of the ports is always port 22 representing the port the ssh server would be listening to. The other port would represent the port the client application would be running on. Due to the relatively small data set

size, the classifiers trained on the other port number. Each service simulation had a different client port number leading to rules or models built around the client port number. Therefore we eliminated port numbers.

We eliminated sequence and acknowledgment numbers as well. This is because, sequence and acknowledgment numbers determine which phase of the connection, the simulation was in. This led to extremely good results when the classifiers were tested with the training data but poor performance when the classifiers were tested on the other data sets.

4.2.1.1 Results

The performance results of the classifiers based on two measures, recall and precision for each ssh service:

Train set	Test set		Precision				Recall			
			<i>LSVM</i>	<i>AdaBoost</i>	<i>RIPPER</i>	<i>J48</i>	<i>LSVM</i>	<i>AdaBoost</i>	<i>RIPPER</i>	<i>J48</i>
<i>inSSHNoPortSyn-1-10000</i>	inSSHNoPortSyn-1-10000	Scp	0.385	0.581	0.782	0.943	0.372	0.568	0.681	0.665
		Sftp	0.5	0.531	0.744	0.747	0.226	0.548	0.848	0.947
		Ssh	0.502	0.748	0.897	0.927	0.674	0.838	0.921	0.942
		xForward	1	0.82	0.94	0.946	0.482	0.644	0.861	0.892
		Tunnel	0.415	0.654	0.898	0.905	0.731	0.711	0.942	0.968
	inSSHNoPortSyn-2-10000	Scp	0.389	0.115	0.23	0.307	0.321	0.052	0.081	0.045
		Sftp	0.419	0.365	0.459	0.478	0.23	0.534	0.784	0.894
		Ssh	0.548	0.834	0.936	0.945	0.668	0.781	0.861	0.879
		xForward	1	0.827	0.905	0.922	0.499	0.768	0.886	0.904
		tunnel	0.469	0.686	0.94	0.916	0.895	0.839	0.98	0.983
	inSSHNoPortSyn-3-10000	scp	0.429	0.514	0.536	0.339	0.44	0.469	0.478	0.108
		sftp	0.549	0.488	0.54	0.479	0.206	0.493	0.638	0.783
		ssh	0.556	0.871	0.929	0.943	0.674	0.838	0.921	0.942
		xForward	1	0.86	0.938	0.915	0.475	0.674	0.825	0.863
		tunnel	0.479	0.669	0.929	0.895	0.916	0.893	0.981	0.99

<i>inSSHNoPortSyn-2-10000</i>	inSSHNoPortSyn-1-10000	scp	0.395	0.223	0.326	0.226	0.119	0.213	0.174	0.116
		sftp	0.375	0.28	0.398	0.438	0.415	0.284	0.423	0.624
		ssh	0.502	0.656	0.815	0.777	0.675	0.793	0.911	0.926
		xForward	1	0.81	0.902	0.855	0.482	0.645	0.719	0.787
		tunnel	0.417	0.61	0.605	0.886	0.738	0.628	0.902	0.844
	inSSHNoPortSyn-2-10000	scp	0.583	0.576	0.808	0.847	0.245	0.757	0.805	0.796
		sftp	0.466	0.678	0.775	0.805	0.426	0.383	0.548	0.861
		ssh	0.548	0.791	0.965	0.972	0.669	0.768	0.935	0.974
		xForward	1	0.903	0.979	0.99	0.499	0.785	0.913	0.936
		tunnel	0.476	0.676	0.709	0.955	0.927	0.867	0.99	0.997
	inSSHNoPortSyn-3-10000	scp	0.371	0.283	0.35	0.318	0.086	0.223	0.219	0.19
		sftp	0.385	0.334	0.394	0.416	0.449	0.368	0.443	0.591
		ssh	0.556	0.801	0.949	0.926	0.675	0.793	0.911	0.926
		xForward	1	0.828	0.931	0.89	0.475	0.694	0.73	0.833
		tunnel	0.478	0.663	0.656	0.944	0.916	0.853	0.991	0.99
<i>inSSHNoPortSyn-3-10000</i>	inSSHNoPortSyn-1-10000	scp	0.396	0.469	0.591	0.507	0.333	0.582	0.555	0.434
		sftp	0.379	0.576	0.598	0.499	0.311	0.323	0.621	0.51
		ssh	0.502	0.652	0.805	0.794	0.675	0.925	0.936	0.939
		xForward	0.972	0.817	0.912	0.801	0.484	0.65	0.853	0.813
		tunnel	0.472	0.609	0.892	0.899	0.708	0.599	0.825	0.832
	inSSHNoPortSyn-2-10000	scp	0.405	0.317	0.204	0.401	0.343	0.278	0.106	0.326
		sftp	0.415	0.334	0.383	0.428	0.31	0.286	0.594	0.471
		ssh	0.548	0.756	0.958	0.954	0.669	0.868	0.898	0.89
		xForward	0.913	0.875	0.958	0.812	0.503	0.774	0.924	0.938
		tunnel	0.53	0.672	0.95	0.956	0.867	0.829	0.978	0.959
	inSSHNoPortSyn-3-10000	scp	0.415	0.501	0.835	0.752	0.355	0.688	0.689	0.953
		sftp	0.408	0.756	0.731	0.923	0.315	0.28	0.866	0.683
		ssh	0.556	0.777	0.964	0.973	0.675	0.925	0.936	0.939
		xForward	0.977	0.847	0.944	0.943	0.477	0.7	0.918	0.938
		tunnel	0.545	0.666	0.945	0.954	0.912	0.826	0.991	0.987

Table 13 – The precision and recall values of the classifiers for each ssh service

Giving an equal importance to precision and recall, we derive a new measure to ease the comparison; F-measure is weighted harmonic mean of precision and recall. The following table gives the average F-measure of all classifiers for every ssh service.

	LSVM	AdaBoost	RIPPER	J48
scp	0.328	0.406	0.458	0.434
sftp	0.357	0.413	0.588	0.625
ssh	0.596	0.796	0.913	0.919
xForward	0.608	0.712	0.885	0.936
tunnel	0.651	0.767	0.888	0.887

Table 14 – Average F-measure of all classifiers for each ssh service

4.2.1.2 Discussion

Table 15 enables us to determine which classifier has a statistically significant better performance than the remaining classifiers. The values in this table represent t-values calculated between the F-measures of the classifier specified on the row heading and the classifier on the column header. The first row, for example, compares the difference of F-measure means between LSVM and AdaBoost, LSVM and RIPPER and finally LSVM and J48.

All classifiers perform poorly on differentiating scp from the remaining ssh services seen by the low recall, precision and F-measure values of all classifiers. There is no statistically significant difference (at a significance level of 0.05) between the performances of all classifiers when differentiating scp from all other services. For the remaining services, all classifiers outperform Linear-SVM. RIPPER and J48 outperform AdaBoost. While J48 slightly outperforms RIPPER, there is no statistically significant difference between both classifiers.

		LSVM	AdaBoost	RIPPER	J48
LSVM	scp		-1.050	-1.343	-0.984
	sftp		-1.653	-4.562	-4.941
	ssh		-13.731	-26.177	-21.660
	xForward		-3.815	-7.978	-14.379
	tunnel		-7.677	-13.745	-13.270
AdaBoost	scp			-0.460	-0.225
	sftp			-3.147	-3.597
	ssh			-6.675	-6.292
	xForward			-4.756	-8.898
	tunnel			-5.383	-5.242
RIPPER	scp				0.178
	sftp				-0.527
	ssh				-0.350
	xForward				-1.552
	tunnel				0.040

Table 15 – t-value comparing each classifier based on the F-measures for each service, a one-tail critical t-value at significance level 0.05 is 1.746

One way of grouping performance measures into an easy measure is to look at the percentage of correctly identified instances. The following table reports the percentage of correctly identified instances for each classifier.

<i>Train set</i>	<i>Test set</i>	% Correct			
		<i>LSVM</i>	<i>AdaBoost</i>	<i>RIPPER</i>	<i>J48</i>
inSSHNoPortSyn-1-10000	inSSHNoPortSyn-1-10000	49.66	66.14	85.04	88.28
	inSSHNoPortSyn-2-10000	52.23	59.45	71.82	74.1
	inSSHNoPortSyn-3-10000	54.19	67.3	76.84	73.7
inSSHNoPortSyn-2-10000	inSSHNoPortSyn-1-10000	48.53	51.24	62.55	65.9
	inSSHNoPortSyn-2-10000	55.29	71.17	83.79	91.26
	inSSHNoPortSyn-3-10000	51.99	58.58	65.86	70.58
inSSHNoPortSyn-3-10000	inSSHNoPortSyn-1-10000	50.18	61.57	75.78	70.54
	inSSHNoPortSyn-2-10000	53.83	60.68	69.99	71.65
	inSSHNoPortSyn-3-10000	54.64	68.37	87.96	89.96

Table 16 – Percentage of instances correctly identified by the different classifiers

Rule-based and decision tree classifiers outperform statistical classifiers. We decided to investigate the reason behind this. We present the rules produced by RIPPER. The decision tree built by J48 has around 479 nodes and 240 leaves. Therefore we cannot easily visualize it on paper. The tree provides a good visualization of how the J48 makes its decisions and could be easily viewed from within Weka.

4.2.1.3 RIPPER rules

RIPPER produces rules that should be followed in sequence. RIPPER produced around 50 rules for each dataset. Since the complete rule set consists of too many rules, we only show rules that cover a 100 or more examples in the training data set.

Tunnel

1. (tcpControlBits <= 16) and
 - o (payloadLength >= 1460) and
 - o (ipTTL >= 127)
 - (804.0/0.0)
2. (payloadLength <= 48) and
 - o (ipTTL >= 128) and
 - o (tcpWindow >= 17424) and

- (interarrivaltime >= 0.005843) and
 - (interarrivaltime <= 0.109102)
 - (264.0/35.0)
3. (payloadLength <= 48) and
- (payloadLength >= 32)
 - (309.0/7.0)
4. (payloadLength <= 16) and
- (ipTTL >= 128) and
 - (tcpWindow >= 17424)
 - (476.0/139.0)

The rules for discriminating tunneling traffic above indicate that ssh tunneling packets either have extremely large payloads or payloads that range from 32 to 48 bytes, see highlighted rules. Inter-arrival times range from 0.005 to 0.1 milliseconds.

sftp

1. (interarrivaltime <= 0.001067) and
 - (tcpWindow <= 16632) and
 - (payloadLength >= 180) and
 - (tcpWindow <= 16512) and
 - (payloadLength >= 564)
 - (170.0/1.0)
2. (interarrivaltime <= 0.001004) and
 - (tcpWindow <= 16632) and
 - (payloadLength >= 180) and
 - (payloadLength <= 232)
 - (211.0/8.0)
3. (interarrivaltime <= 0.009179) and
 - (ipTTL >= 127) and
 - (tcpWindow >= 17264) and
 - (payloadLength >= 180)
 - (240.0/30.0)
4. (interarrivaltime <= 0.010969) and
 - (payloadLength <= 28) and
 - (ipTTL >= 127) and
 - (tcpWindow >= 17520)
 - (816.0/402.0)
5. (interarrivaltime <= 0.010969) and
 - (ipTTL >= 127) and
 - (payloadLength >= 564) and
 - (tcpWindow <= 16632)
 - (132.0/9.0)
6. (interarrivaltime <= 0.011868) and
 - (ipTTL >= 127) and
 - (ipTTL <= 127) and
 - (interarrivaltime >= 0.000018) and
 - (interarrivaltime <= 0.000438)
 - (258.0/116.0)

The inter-arrival times for sftp have much smaller ranges, even between 0.000018 and 0.000438 milliseconds and payload lengths that are usually larger than 564 bytes.

xForward

1. (ipTTL <= 63)
 - (961.0/0.0)
2. (ipTTL >= 128) and
 - (payloadLength <= 116) and
 - (payloadLength >= 84) and
 - (interarrivaltime <= 0.013233)
 - (311.0/24.0)
3. (ipTTL >= 128) and
 - (tcpWindow <= 16644) and
 - (payloadLength >= 276)
 - (108.0/7.0)
4. (ipTTL >= 128) and
 - (payloadLength <= 116) and
 - (tcpControlBits <= 16) and
 - (interarrivaltime >= 0.101755)
 - (100.0/32.0)

This result shows the classifier's ability to detect anomalous behavior really well and build rules around it. X-Forwarding was the only ssh service, we could not simulate on our local network and therefore used a public server. By examining time to live, a measure that reflects the number of hops between two machines, the classifier is capable of accurately determining x-forwarding traffic from the remaining ssh services.

ssh

1. (interarrivaltime >= 0.013664) and
 - (payloadLength <= 52) and
 - (payloadLength >= 52) and
 - (ipTTL <= 127)
 - (680.0/6.0)
2. (interarrivaltime >= 0.013664) and
 - (interarrivaltime >= 0.092043) and
 - (tcpWindow <= 16964) and
 - (interarrivaltime <= 0.16929)
 - (129.0/7.0)
3. (interarrivaltime >= 0.009987) and
 - (interarrivaltime >= 0.171519)
 - (544.0/70.0)
4. (payloadLength <= 164) and
 - (interarrivaltime >= 0.009993) and
 - (payloadLength >= 116) and
 - (ipTTL >= 128)
 - (157.0/13.0)
5. (payloadLength <= 164) and

- (interarrivaltime >= 0.012093) and
- (interarrivaltime >= 0.068168) and
- (interarrivaltime <= 0.167966)
 - (124.0/28.0)

6. (payloadLength <= 164) and

- (interarrivaltime <= 0.000044)
 - (128.0/15.0)

7. (payloadLength <= 164) and

- (interarrivaltime >= 0.010149) and
- (payloadLength <= 52) and
- (tcpWindow <= 17156) and (interarrivaltime <= 0.066476)
 - (114.0/36.0)

For interactive shell, commands sent from the client to the server do not result in large payload lengths. Server responses are generally not large. This is reflected by the bounds placed on the number of bytes. There are also several rules indicating a large inter-arrival time, which is probably related to user reaction or response times.

4.2.2 Experiment 2: Effect of reducing features on the performance of RIPPER in differentiating ssh services

RIPPER was capable of determining that x-Forwarding by identifying its an unusual hop count. xForwarding was not simulated within our local network where all other services were simulated. This caused the classifiers to have high recall and precision values for x-Forwarding.

By eliminating several features gradually, we evaluated the performance of RIPPER with different features selected. We report the results of this experiment and present the rules learned by RIPPER with only 3 features selected.

The feature sets are as follows:

5-Feature set:

1. Inter-arrival time
2. Time to live
3. TCP Control bits
4. TCP Window

5. Payload length

4-Feature set:

1. Inter-arrival time
2. TCP Control bits
3. TCP Window
4. Payload length

3-Feature set:

1. Inter-arrival time
2. TCP Control bits
3. Payload length

4.2.2.1 Results

Train set	Test set		Precision			Recall			F-measure		
			<i>RIPPER-3</i>	<i>RIPPER-4</i>	<i>RIPPER-5</i>	<i>RIPPER-3</i>	<i>RIPPER-4</i>	<i>RIPPER-5</i>	<i>RIPPER-3</i>	<i>RIPPER-4</i>	<i>RIPPER-5</i>
<i>inSSHxFeatures-1-10000</i>	inSSHx Feature s-1- 10000	scp	0.654	0.659	0.789	0.084	0.530	0.568	0.149	0.588	0.661
		sftp	0.557	0.782	0.815	0.422	0.495	0.732	0.480	0.606	0.771
		ssh	0.850	0.885	0.911	0.810	0.863	0.893	0.830	0.874	0.902
		xForward	0.346	0.938	0.644	0.749	0.784	0.889	0.473	0.854	0.747
		Tunnel	0.771	0.548	0.902	0.768	0.961	0.922	0.769	0.698	0.912
	inSSHx Feature s-2- 10000	scp	0.589	0.206	0.390	0.073	0.085	0.117	0.130	0.120	0.180
		sftp	0.537	0.380	0.469	0.428	0.424	0.707	0.476	0.401	0.564
		ssh	0.828	0.945	0.906	0.766	0.799	0.849	0.796	0.866	0.877
		xForward	0.359	0.987	0.708	0.770	0.855	0.878	0.490	0.916	0.784
		tunnel	0.827	0.554	0.938	0.836	0.976	0.954	0.831	0.707	0.946
	inSSHx Feature s-3- 10000	scp	0.654	0.564	0.507	0.075	0.437	0.175	0.135	0.492	0.260
		sftp	0.512	0.625	0.525	0.385	0.399	0.656	0.440	0.487	0.583
		ssh	0.852	0.944	0.917	0.810	0.863	0.893	0.830	0.902	0.905
		xForward	0.354	0.942	0.639	0.761	0.813	0.880	0.483	0.873	0.740
		tunnel	0.828	0.540	0.918	0.859	0.978	0.967	0.843	0.696	0.942

<i>inSSHxFeatures-2-10000</i>	inSSHx Feature s-1- 10000	scp	0.341	0.196	0.216	0.489	0.086	0.117	0.402	0.120	0.152
		sftp	0.527	0.345	0.417	0.484	0.348	0.498	0.505	0.346	0.454
		ssh	0.785	0.425	0.778	0.790	0.954	0.924	0.787	0.588	0.845
		xForward	0.661	0.930	0.923	0.449	0.658	0.703	0.535	0.771	0.798
		tunnel	0.749	0.815	0.628	0.719	0.491	0.828	0.734	0.613	0.714
	inSSHx Feature s-2- 10000	scp	0.416	0.854	0.821	0.542	0.684	0.771	0.471	0.760	0.795
		sftp	0.554	0.837	0.786	0.530	0.456	0.680	0.542	0.590	0.729
		ssh	0.853	0.538	0.972	0.825	0.982	0.946	0.839	0.695	0.959
		xForward	0.856	0.990	0.986	0.634	0.855	0.907	0.728	0.918	0.945
		tunnel	0.833	0.910	0.759	0.863	0.881	0.990	0.848	0.895	0.859
	inSSHx Feature s-3- 10000	scp	0.362	0.281	0.259	0.552	0.108	0.156	0.437	0.156	0.195
		sftp	0.505	0.356	0.394	0.449	0.381	0.463	0.475	0.368	0.426
		ssh	0.856	0.487	0.939	0.790	0.954	0.924	0.822	0.645	0.931
		xForward	0.760	0.944	0.936	0.472	0.687	0.735	0.582	0.795	0.823
		tunnel	0.832	0.898	0.626	0.868	0.770	0.912	0.850	0.829	0.742
<i>inSSHxFeatures-3-10000</i>	inSSHx Feature s-1- 10000	scp	0.553	0.557	0.565	0.241	0.309	0.614	0.336	0.397	0.588
		sftp	0.430	0.442	0.632	0.731	0.710	0.560	0.541	0.545	0.594
		ssh	0.703	0.751	0.816	0.912	0.926	0.927	0.794	0.829	0.868
		xForward	0.734	0.930	0.889	0.494	0.768	0.857	0.591	0.841	0.873
		tunnel	0.766	0.884	0.894	0.687	0.691	0.830	0.724	0.776	0.861
	inSSHx Feature s-2- 10000	scp	0.582	0.269	0.263	0.255	0.115	0.171	0.355	0.161	0.207
		sftp	0.422	0.370	0.396	0.741	0.663	0.549	0.538	0.475	0.460
		ssh	0.807	0.861	0.950	0.850	0.863	0.881	0.828	0.862	0.914
		xForward	0.835	0.936	0.920	0.637	0.824	0.927	0.723	0.876	0.923
		tunnel	0.843	0.916	0.950	0.835	0.824	0.978	0.839	0.868	0.964
	inSSHx Feature s-3- 10000	scp	0.649	0.810	0.649	0.263	0.619	0.766	0.374	0.702	0.703
		sftp	0.451	0.610	0.725	0.787	0.832	0.589	0.573	0.704	0.650
		ssh	0.847	0.894	0.947	0.912	0.926	0.927	0.878	0.910	0.937
		xForward	0.834	0.963	0.937	0.652	0.847	0.920	0.732	0.901	0.928
		tunnel	0.863	0.936	0.945	0.859	0.898	0.991	0.861	0.917	0.967

Table 17 – Precision, recall and F-measure values of RIPPER with different features

Train set	Test set	RIPPER-3	RIPPER-4	RIPPER-5
inSSHxFeatures-1-10000	inSSHxFeatures-1-10000	56.65	72.62	80.05
	inSSHxFeatures-2-10000	57.45	62.78	70.07
	inSSHxFeatures-3-10000	57.77	69.76	71.41
inSSHxFeatures-2-10000	inSSHxFeatures-1-10000	58.61	50.72	61.37
	inSSHxFeatures-2-10000	67.83	77.14	85.86
	inSSHxFeatures-3-10000	62.61	57.98	63.76
inSSHxFeatures-3-10000	inSSHxFeatures-1-10000	61.28	68.06	75.74
	inSSHxFeatures-2-10000	66.33	65.77	70.1
	inSSHxFeatures-3-10000	69.42	82.41	83.85

Table 18 – Percentage of instances correctly identified by RIPPER when using different features

With as low as four features, RIPPER was capable of producing an average of 67% correctly identified instances. While RIPPER with four features performs worse than with five features, we expect it to give more accurate results on public data sets as it is independent of the unusual hop count value, which RIPPER used to identify xForwarding.

The rules learned by RIPPER with 3 features are as follows:

tunnel

1. (tcpControlBits <= 16) and
 - (payloadLength >= 1460)
 - (1021.0/217.0)
2. (payloadLength <= 48) and
 - (payloadLength >= 32)
 - (318.0/8.0)
3. (tcpControlBits <= 16) and
 - (interarrivaltime >= 0.001194) and
 - (interarrivaltime <= 0.002863) and
 - (interarrivaltime <= 0.00176)
 - (272.0/98.0)
4. (tcpControlBits <= 16) and
 - (interarrivaltime >= 0.002251) and
 - (interarrivaltime <= 0.002863) and
 - (interarrivaltime >= 0.002446)
 - (148.0/65.0)

sftp

1. (interarrivaltime <= 0.001004) and
 - (payloadLength >= 180) and
 - (payloadLength >= 564)
 - (836.0/363.0)
2. (interarrivaltime <= 0.008516) and
 - (interarrivaltime <= 0.000712) and
 - (interarrivaltime >= 0.000009) and
 - (interarrivaltime <= 0.000031)
 - (678.0/307.0)

scp

1. (interarrivaltime <= 0.007801) and
 - (payloadLength >= 180) and
 - (payloadLength >= 616) and
 - (interarrivaltime <= 0.002028)
 - (207.0/70.0)

ssh

1. (interarrivaltime >= 0.011136) and
 - (payloadLength <= 52) and
 - (payloadLength >= 52)
 - (1111.0/77.0)
2. (interarrivaltime >= 0.011136) and
 - (interarrivaltime >= 0.134065) and
 - (tcpControlBits <= 16) and

- (interarrivaltime <= 0.233085)
 - (324.0/99.0)
- 3. (interarrivaltime >= 0.011136) and
 - (interarrivaltime <= 0.047156) and
 - (payloadLength >= 68) and
 - (payloadLength <= 164) and
 - (payloadLength >= 148)
 - (116.0/7.0)

5 Future work

Our four month investigatory study into the success of using machine learning algorithms to classify ssh and its different services within network traffic data showed very promising results. The following is a non-exhaustive list of the different ideas we would wish to examine in the future:

- The performance gain of using non-linear SVMs as opposed to linear SVMs.
- The performance gain of using flow-based data as opposed to packet-based data.
- The performance gain of using payload information; we suggest using term-document matrices to deal with words in payload and feature selection algorithms to limit the number of terms we use from payload. An alternative approach would be to look at the contents of the first 64 bytes of payload and represent them in a format suitable for classification.
- Building a temporal model of classifiers that trains on several packets in sequence: each attribute would have a separate classifier that determines the type of application based only on that attribute. These first layer classifiers would be trained on instances, each containing x number of packets, t packets apart. A second layer classifier would then take a vote from all first layer classifiers on the type of application running.

6 Conclusion

Linear-SVM performs poorly compared to rule based classifiers. We believe this is mainly due to network traffic data being non-linearly separable. Our investigatory

study shows that rule-based classifiers like RIPPER significantly outperform other classifiers, where the t-tests are employed at 95% confidence level. A 2-phase system that first classifies ssh traffic and then classifies the type of service ssh is running is plausible. Using only packet-based classification, this system achieves a peak performance of 100% detection of ssh traffic and 82% detection of the type of service running on ssh. There are several limitations to our classifiers, mainly they require training data sets specific to the network they are to classify traffic. A more generic classifier would require larger data sets that cover a wider variety of application scenarios and perhaps a change of paradigm from packet-based to flow-based classifiers as well as the use of temporal features.

7 References

- [1] D. J. Barrett and R. E. Silverman, SSH, the Secure Shell: The Definitive Guide. Sebastopol, CA, USA: O'Reilly & Associates, Inc, 2001, ISBN: 0596008953.
- [2] W. W. Cohen, "Fast effective rule induction." in ICML, 1995, pp. 115-123.
- [3] G. Combs. (2006, Ethereal. 0.99.0, Available: <http://www.ethereal.com/>)
- [4] T. Dean, Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99, Stockholm, Sweden, July 31 - August 6, 1999. 2 Volumes, 1450 Pages. Morgan Kaufmann, 1999.
- [5] Y. Freund and R. E. Schapire, "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting." J. Comput. Syst. Sci., vol. 55, pp. 119-139, 1997.
- [6] K. Fujii. (2006, April). Jpcap - java package for packet capture. 0.5.1, Available: <http://netresearch.ics.uci.edu/kfujii/jpcap/doc/index.html>
- [7] P. Haffner, S. Sen, O. Spatscheck and D. Wang, "ACAS: Automated construction of application signatures," in MineNet '05: Proceeding of the 2005 ACM SIGCOMM Workshop on Mining Network Data, 2005, pp. 197-202.
- [8] M. A. Hearst, "Trends Controversies: Support Vector Machines," IEEE Intelligent System, vol. 13, pp. 18-28, 1998.
- [9] A. Hussain, G. Bartlett, Y. Pryadkin, J. Heidemann, C. Papadopoulos and J. Bannister, "Experiences with a continuous network tracing infrastructure," in Proceedings of the {ACM} SIGCOMM MineNet Workshop, 2006, pp. 190.
- [10] A. Hussain, G. Bartlett, Y. Pryadkin, J. Heidemann, C. Papadopoulos and J. Bannister, "Experiences with a continuous network tracing infrastructure," in MineNet '05: Proceeding of the 2005 ACM SIGCOMM Workshop on Mining Network Data, 2005, pp. 185-190.
- [11] H. Liu and R. Setiono, "Chi2: Feature Selection and Discretization of Numeric Attributes," Ictai, vol. 00, pp. 388, 1995.
- [12] A. W. Moore and K. Papagiannaki, "Toward the accurate identification of network applications." in PAM, 2005, pp. 41-54, Available: <http://sprngernk.metapress.com/openur.asp?genre=arte&ssn=0302-9743&oume=3431&spage=41>.

- [13] A. W. Moore and D. Zuev, "Internet traffic classification using bayesian analysis techniques." in SIGMETRICS, 2005, pp. 50-60, Available: <http://do.am.org/10.1145/1064212.1064220>.
- [14] J. Postel. (1981, sep). Internet protocol. Available: <http://www.ietf.org/rfc/rfc791.txt>;
- [15] J. Postel. (1981, sep). Transmission control protocol. Available: <http://www.rfc-editor.org/rfc/rfc793.txt>
- [16] J. Postel, (1980, aug). User datagram protocol. Available: <http://www.rfc-editor.org/rfc/rfc768.txt>
- [17] R. E. Schapire, "A brief introduction to boosting." in IJCAI, 1999, pp. 1401-1406.
- [18] I. H. Witten and E. Frank, Data Mining: Practical Machine Learning Tools and Techniques. ,Second ed.Morgan Kaufmann, 2005, ISBN: 1558605525.
- [19] C. Wright, F. Monroe and G. M. Masson, "HMM profiles for network traffic classification," in VizSEC/DMSEC '04: Proceedings of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security, 2004, pp. 9-15.

Appendix A

Extract from the Weka source code documentation of JRip, an implementation of

RIPPER:

The algorithm is briefly described as follows:

Initialize $RS = \{\}$, and for each class from the less prevalent one to the more frequent one, DO:

1. Building stage: repeat 1.1 and 1.2 until the description length (DL) of the ruleset and examples is 64 bits greater than the smallest DL met so far, or there are no positive examples, or the error rate $\geq 50\%$.

1.1. Grow phase:

Grow one rule by greedily adding antecedents (or conditions) to the rule until the rule is perfect (i.e. 100% accurate). The procedure tries every possible value of each attribute and selects the condition with highest information gain: $p(\log(p/t) - \log(P/T))$.

1.2. Prune phase:

Incrementally prune each rule and allow the pruning of any final sequences of the antecedents;

The pruning metric is $(p-n)/(p+n)$ -- but it's actually $2p/(p+n) - 1$, so in this implementation we simply use $p/(p+n)$ (actually $(p+1)/(p+n+2)$, thus if $p+n$ is 0, it's 0.5).

2. Optimization stage: after generating the initial ruleset $\{R_i\}$, generate and prune two variants of each rule R_i from randomized data using procedure 1.1 and 1.2. But one variant is generated from an empty rule while the other is generated by greedily adding antecedents to the original rule. Moreover, the pruning metric used here is $(TP+TN)/(P+N)$.

Then the smallest possible DL for each variant and the original rule is computed. The variant with the minimal DL is selected as the final representative of R_i in the ruleset. After all the rules in $\{R_i\}$ have been examined and if there are still residual positives, more rules are generated based on the residual positives using Building Stage again.

3. Delete the rules from the ruleset that would increase the DL of the whole ruleset if it were in it. and add resultant ruleset to RS .

ENDDO

Appendix B

The following pages contain detailed information on the different public data set properties.