A Comparison of Unsupervised Learning Techniques for Encrypted Traffic Identification

Carlos Bacquet[1], Kubra Gumus[2], Dogukan Tizer[3], A. Nur Zincir-Heywood[4] and Malcolm I. Heywood[5]

[1]Faculty of Computer Science, Dalhousie University,
6050 University Avenue, Halifax NS, Canada
bacquet@cs.dal.ca

[2]Computer Engineering Department, Bilkent University,
06800 Bilkent, Ankara, Turkey
k_gumus@ug.bilkent.edu.tr

[3]Computer Engineering Department, Ege University,
35040 Bornova-IZMIR, Turkey
05050007004@ogrenci.ege.edu.tr

[4]Faculty of Computer Science, Dalhousie University,
6050 University Avenue, Halifax NS, Canada
zincir@cs.dal.ca

[5]Faculty of Computer Science, Dalhousie University,
6050 University Avenue, Halifax NS, Canada
mheywood@cs.dal.ca

***Abstract*: The increasing use of encrypted traffic combined with non-standard port associations makes the task of traffic identification increasingly difficult. This work benchmarks the performance of five unsupervised clustering algorithms: Basic K-Means, Semi-supervised K-Means, DBSCAN, EM, and MOGA for encrypted traffic identification, specifically SSH. Results show that the performance of MOGA, a multi objective clustering approach using a Genetic Algorithm, is not only better than the others, but also provides a good trade off in terms of detection rate, false positive rate, and time to built and run the model. This is a very desirable property for a potential implementation of an encrypted traffic identification system.**

***Keywords*: Unsupervised Machine Learning, Clustering Algorithms, Genetic Algorithms, K-Means, DBSCAN, EM, Encrypted Traffic Identification.

1. Introduction

An important part of network management requires the accurate identification and classification of network traffic [4], [2]. Network administrators are normally interested in identifying application types for decisions regarding both bandwidth management and quality of service [2]. A particularly interesting area in network traffic identification pertains to encrypted traffic, where the fact that the payload is encrypted represents an additional degree of uncertainty. Specifically, many traditional approaches to traffic classification rely on payload inspection, which become unfeasible under packet encryption [12], [20], [7], [6], [18]. An alternative to payload inspection would be the use of port numbers to identify application types. However, this practice has become increasingly inaccurate, as users are now able to arbitrarily change the port number to deceive security mechanisms [3], [20], [7], [18], [6]. In short, the traditional approaches are unable to deal with the identification of encrypted traffic.

In this work we benchmark several unsupervised learning algorithms in the identification of encrypted traffic, where Secure Shell (SSH) is chosen as an example encrypted application. While SSH is typically used to remotely access a computer, it can also be utilized for "tunneling, file transfers and forwarding arbitrary TCP ports over a secure channel between a local and a remote computer" [2]. These properties of SSH make it an interesting encrypted application to focus on, given that it shows similar behavior like popular encrypted applications such as Skype. However, unlike Skype, SSH is an open source protocol. This ensures that the ground truth is known regarding the traffic tested. From the traffic identification perspective we benchmark four unsupervised clustering techniques: basic K-Means, semi-supervised K-Means, DBSCAN, and EM; and compare the results with a Multi-Objective Genetic Algorithm (MOGA) that is used for the dual identification of appropriate (flow) feature subspace and clustering of traffic types. We first proposed MOGA in

[5], and it assumes that the resulting clusters partition traffic into encrypted/not encrypted. Results show that MOGA provides 93.5% detection rate and 0.7% false positive with very fast building and run time, outperforming all the other models presented here.

2. Previous Work

Given the limitations of port number analysis and payload inspection, several previous attempts to identify encrypted traffic have worked with statistics based on flows [2], [20], [7]. A number of these attempts have employed supervised learning methods. However, these classifiers have uncertain generalization properties when faced with new data [20], [7]. One alternative to classifiers is the use of clustering mechanisms or unsupervised learning methods. The following is an analysis of previous work with unsupervised methods.

Recently, Siqueira et al. presented a clustering approach to identify Peer-to-Peer (P2P) versus non-P2P in TCP traffic [12]. A total of 249 features from the headers of the packets were considered, out of which the best 5 were selected to parameterize clusters. The selection of the features was based on the variance of their values, such that the higher the variance, the better the perceived discrimination of a feature. This methodology enabled the authors to achieve results of 86.12% detection rate for P2P applications with an average accuracy of 96.79%. Their proposed method utilized the port number as one of the five selected features.

Erman et al. [8] compared an unsupervised approach using an Expectation Maximization (EM) based clustering algorithm (AutoClass), against a supervised approach that used a Naive Bayes Classifier. Both methods were tested on subsets of the traces Auckland IV and Auckland VI. Their results showed that the unsupervised technique had an accuracy of up to 91%, outperforming the supervised technique by up to 9%. Furthermore, they found that the unsupervised technique was also able to discover traffic from previously unknown applications. In [7], they presented an evaluation of three clustering algorithms: K-Means, DBSCAN and AutoClass. The authors used two datasets, one of them (Auckland IV) containing DNS, FTP, HTTP, IRC, LimeWire, NNTP, POP3, and SOCKS; and the other (Calgary trace), containing HTTP, P2P, SMTP, and POP3. The authors found that with K-Means the overall accuracy steadily improved as the number of clusters was increased. This continued until K was around 100 with the overall accuracy being 79% and 84% on each data set respectively. Thus, from their results they observed that K-Means seemed to provide the best mix of properties. Then in [9], they presented a semi-supervised classification technique. The proposed model achieved high flow and byte accuracy (greater than 90%), with only a few labelled flows and many unlabeled flows. The authors employed a K-Means algorithm, and the employed traces were collected from the University of Calgary Internet link.

Bernaille et al. used information from the first five packets of a TCP connection to identify applications [6]. In particular, they analyzed the size of the first few packets, which captures the application's negotiation phase. They then used a K-Means algorithm to cluster these features during the learning phase. After several experiments they concluded that the best K number of clusters was 50, achieving results of up to 96.9% accuracy with SSH. The authors mention that a potential limitation could be the fact that the method is sensitive to packet order.

Yingqiu et al. also presented a flow based clustering approach, using K-Means to build the clusters with features previously identified as the best discriminators [20]. To this end, several feature selection and search techniques are performed. After clustering, the authors applied a log transformation, which enabled them to reach an overall accuracy level of up to 90% when utilizing K= 80 clusters. The authors concluded this was a very promising approach for TCP classification.

To the best of our knowledge, this is the first work that benchmarks a genetic algorithm for the dual problem of feature selection and clustering, to the performance of the aforementioned unsupervised learning techniques for encrypted traffic identification.

3. Methodology

In this section we first characterize the data utilized for the training and testing. This is followed by a description of the process of generating flows from the data source. We then explain of the four clustering techniques to be evaluated: basic K-Means, semi-supervised K-Means, DBSCAN, and EM; and we finish with a description of our proposed model: MOGA for combined feature subspace selection and clustering.

3.1 Data Set

The data set used was captured by the Dalhousie University Computing and Information Services Centre (UCIS) in January 2007 on the campus network between the university and the commercial Internet. Dalhousie University is one of he biggest universities in the Atlantic region of Canada, with more than 15,000 students and about 3300 faculty and staff. Given the privacy related issues the university may face, the data was filtered to scramble the IP addresses and each packet was further truncated to the end of the IP header so that all the payload was excluded. Furthermore, the checksums were set to zero since they could conceivably leak information from short packets. However, any length information in the packet was left intact. Dalhousie traces were labeled by a commercial classification tool called PacketShaper, i.e., a deep packet analyzer [16]. PacketShaper uses Layer 7 filters (L7) to classify the applications. Given that the handshake part of SSH protocol is not encrypted, we can confidently assume that the labeling of the data set is 100% accurate and provides the ground truth for our data set. We emphasize that our work did not consider any information from the handshake phase nor any part of the payload, IP addresses or port numbers. The handshake phase was used solely to obtain the ground truth to which we compare our obtained results.

Also, we focus on SSH as a case study, we could have employed any other encrypted traffic protocol. However, the fact that the SSH's handshake is not encrypted, allowed us to compare our obtained results with those obtained through payload inspection. In order to build training data we sampled the data set. The training data for all the clustering algorithms and for the MOGA, unless stated otherwise, consisted of 12250 flows, including SSH, MSN, HTTP, FTP, and DNS. The test data, on the other hand, was the entire data set (more than 18,500,000 flows) and consisted of flows of each of those applications, plus flows that belonged to any of the following applications: RMCP, Oracle SQL*NET, NPP, POP3, NETBIOS Name Service, IMAP, SNMP, LDAP, NCP, RTSP, IMAPS and POP3S.

3.2 Flow Generation

Flows are defined by sequences of packets that present the same values for source IP address, destination IP address, source port, destination port and type of protocol [12]. Each flow is described by a set of statistical features and associated feature values. A feature is a descriptive statistic that can be calculated from one or more packets. We used the NetMate [15] tool set to process data sets, generate flows, and compute feature values. In total, 38 features were obtained from NetMate (table 1). Flows are bidirectional with the first packet determining the forward direction. Since flows are of limited duration, in this work UDP flows are terminated by a flow timeout, and TCP flows are terminated upon proper connection teardown or by a flow timeout, whichever occurs first. A 600 second flow timeout value was employed here; where this corresponds to the IETF Realtime Traffic Flow Measurement working groups architecture [11]. It is important to mention that only UDP and TCP flows are considered. Specifically, flows that have no less than one packet in each direction, and transport no less than one byte of payload. Payload data and features like IP addresses and source/destination port numbers were excluded from the feature set to ensure that the results were not dependent on such biases.

In this work, we employed the same features used in [9] for the basic K-Means, semi-supervised K-Means, DBSCAN, and EM algorithms, table 2. For the MOGA, on the other hand, we employed all the 38 original features for it to select the most appropriate ones during the training phase, table 1. In order to reproduce the features employed in [9] we modified the *netai_flowstats.c* module of NetMate, originally used to obtained the aforementioned 38 features employed by MOGA. The reason we employed the exact same features as in [9] was to accurately reproduce the methodology there described.

3.3 Unsupervised Learning Algorithms

The use of clustering algorithms for traffic classification is normally done in two phases. The first phase consists of training the model with a relatively small set of data (training data), and the second phase consists of using the trained model to classify unknown traffic. During the training phase, the training data is used to build clusters based on some criteria of similarity, which will ideally separate the data into similar clusters (groups). The resulting clusters need then to be labelled, which is normally based on the class of the majority of the flows in each cluster, if these clusters will be used for identifying traffic. The second phase consist of assigning a class to the flows to be identified, depending on the label of the cluster that each flow is more similar to. For the experiments presented here, this criteria of similarity will be the Euclidean distance, defined by:

$$d(p,q) = \sqrt{\sum_{i=0}^{n}(pi - qi)^2}$$

The three clustering algorithms selected for this work are K-Means, DBSCAN, and EM. The selection of these algorithms is based in part on the work of Erman et. al in [7], in which the authors made a similar comparison. Furthermore, we also compared these systems to a semi-supervised

Table 1: Features obtained from Netmate

ind.	Feature Name	Abreviation
1	protocol (tcp, udp)	*proto*
2	total forward packets	*total_fpackets*
3	total forward volume	*total_fvolume*
4	total backward packets	*total_bpackets*
5	total backward volume	*total_bvolume*
6	min forward packet length	*min_fpktl*
7	mean forward packet length	*mean_fpktl*
8	max forward packet length	*max_fpktl*
9	std dev forward packet length	*std_fpktl*
10	min backward packet length	*min_bpktl*
11	mean backward packet length	*mean_bpktl*
12	max backward packet length	*max_bpktl*
13	std dev backward packet length	*std_bpktl*
14	min forward inter arrival time	*min_fiat*
15	mean forward inter arrival time	*mean_fiat*
16	max forward inter arrival time	*max_fiat*
17	std dev forward inter arrival time	*std_fiat*
18	min backward inter arrival time	*min_biat*
19	mean backward inter arrival time	*mean_biat*
20	max backward inter arrival time	*max_biat*
21	std dev backward inter arrival time	*std_biat*
22	duration of the flow	*duration*
23	min active	*min_active*
24	mean active	*mean_active*
25	max active	*max_active*
26	std dev active	*std_active*
27	min idle	*min_idle*
28	mean idle	*mean_idle*
29	max idle	*max_idle*
30	std dev idle	*std_idle*
31	sub flow forward packets	*sflow_fpackets*
32	sub flow forward bytes	*sflow_fbytes*
33	sub flow backward packets	*sflow_bpackets*
34	sub flow backward bytes	*sflow_bbytes*
35	forward push counter	*fpsh_cnt*
36	backward push counter	*bpsh_cnt*
37	forward urg counter	*furg_cnt*
38	backward urg counter	*burg_cnt*

method proposed in [9]. Then, we compare the performance of all these algorithms to the performance of the MOGA we first proposed in [5]. The following subsections give a brief explanation of these algorithms, more details can be found in [7] and [9]. For the implementations of these algorithms we used the K-Mean, DBSCAN, and EM provided by Weka [17].

3.3.1 K-Means

K-Means clustering is a method of unsupervised learning, which aims to partition *n* observations into *K* clusters in where each observation belongs to the cluster with the nearest mean. Erman et al. observed in their experiments that one of the main advantages of the K-Means algorithm over other clustering algorithms was the resulting clusters tended to be mainly of a single application type [7]. In our experiment we tested several values of *K* (20, 40, 60, 80, 100, 200, 300, 400), and selected the best result to be compared with the other models. A more detailed explanation of the algorithm can be found in [1].

3.3.2 Semi-supervised K-Means

We followed the semi-supervised approach proposed in [9], in which high detection rates are achieved by labeling the resulting clusters with only a small fraction of the training data labeled. For this semi-supervised approach, we first trained the model with only 5% of the original training data using K-Means clustering. Then we labeled the clusters post training where only 613 out of 12250 flows were labeled. We also trained with larger data sets consisting of 32000 flows, with 80, 800, and 8000 of them labeled. Following the work in [9], the *K* number of clusters was set to 400. In all of these experiments the total number of SSH flows was 6000, so that we had a base point of comparison with the other methods presented. In addition, we also evaluated the effectiveness of the weighted sampling approach proposed in [9]. We generated a 3602 flows training data, out of which only 180 flows were labeled. For this approach we selected 50% of the flows from below, and 50% of the flows above the 95th percentile of the flow transfer size of our original training data (12250 flows).

Table 2: Features for Clustering Algorithms

Total Number of Packets= fpackets+bpackets
Total Caller to Calle Payload Bytes= fvolume-fhlen
Total Bytes= fvolume+bvolume
Total Caller to Callee Header Bytes= fhlen
Total Header(Transport + Network Layer)= fhlen+bhlen
Number of Callee to Caller Packets= bpacket
Average Packet Size= (fpktl + bpktl)/(fpackets + bpackets)
Total Callee to Caller Payload Bytes= bvolume-bhlen
Number of Caller to Callee Packets= fpacket
Total Callee to Caller Header Bytes= bhlen
Total Caller to Callee Bytes= fvolume

3.3.3 DBSCAN

DBSCAN [10] is a density based algorithm, so it regards “clusters as dense areas of objects that are separated by less dense areas” [7]. The main advantage of this algorithms is that unlike K-Means, it is not limited to “spherical shaped clusters but can find clusters of arbitrary shapes” [7]. The DBSCAN algorithm takes two input parameters, epsilon (*eps*) and the number of minimum points (*minPts*). *minPts* is the minimum required points to form a core object, and *eps* is the distance between two objects to be considered “eps-neighbors”. DBSCAN does not take as an input the number of clusters to generate, it finds the optimum number of clusters based on the *minPts* and *eps*. Also, unlike K-means and EM, if an object is not part of an existing cluster it is considered noise [7]. Yang et al. provide a complete description of DBSCAN in [19].

3.3.4 EM

The Expectation Maximization algorithm works with the probabilities that each instance belong to each cluster [17]. The algorithm has two phases, an expectation phase and a maximization phase. The parameters used by the algorithm that “govern the distinct probability distribution of each cluster” [7] are estimated during the expectation phase, and are continually re-estimated during the maximization phase [7]. A more detailed explanation of the algorithm can be found in [1].

3.4 Proposed Algorithm

In this section we provide a detailed explanation of the MOGA for combined feature subspace selection and clustering. We also describe the post-training phase, used to determine which of a Pareto front of non-dominated solutions is selected as the best individual and final solution. The entire process is outlined in figures 1 and 2.

3.4.1 Feature Selection

Feature selection is “the process of choosing a subset of the original predictive variables by eliminating redundant and uninformative ones” [13]. By using a smaller number of features we save significant computing time, and “often build models that generalize better to unseen points” [13]. In this work we took the feature selection model proposed by YeongSeog et al. [13], and modified its evolutionary component to follow the model proposed by Kumar et al. [14]. The latter ensures a convergence towards the Pareto-front (set of non-dominated solutions) without any complex sharing/niching mechanism. One specific property of this Genetic Algorithm (GA) is the utility of a steady-state GA, thus only one or two members of the population are replaced at a time, figure 2.

3.4.2 GA for Feature Selection and Clustering

A GA starts with a population of individuals (potential solutions to a problem), and incrementally evolves that population into better individuals, as established by the fitness criteria. Fitness is naturally relative to the population. Then, for several iterations, individuals are selected to be combined (crossover) to create new individuals (offspring) under a fitness proportional selection operator. In order to model the problem of feature selection to the GA, each individual in the population represents a subset of features *f* and a number of clusters *K*. Specifically, an individual is a 60 bit binary string (100 bit string for our second set of experiments), where bits between the first bit and the 38th bit represent the features to include, and the remaining 21 bits represent the *K* number of clusters (remaining 61 for the second set of experiments). The zeroth bit is related to the port number so it is always ignored. Bits of the individuals in the initial population are initialized with a uniform probability distribution. For feature selection, a one implies to include the feature at that index (from table1), and a zero means to discard it. The *K* number of clusters, on the other hand, is represented by the number of "ones" (as opposed to "zeros") contained between the 39th bit and the 59th bit, plus two (between the 39th bit and the 99th bit for second set of experiments). The reason for adding two is that zero or one cluster would never be a solution. Clusters are identified using the standard K-means algorithm, using that subset of features *f*, and the number of clusters *K*, as the input for the K-means algorithm. Like on the previously discussed algorithms, we used the K-means algorithm provided by Weka [17]. The fitness of the individual will then depend on how well the resulting clusters perform in relation to the following four predefined clustering objectives:

-*Fwithin* (measures cluster cohesiveness, the more cohesive the better). We calculate the average standard deviation per cluster. That is, per each i'th cluster, the sum of the standard deviations per feature over the total number of employed features. Then *Fwithin* will be *K* over the sum of all the clusters' average standards.

-*Fbetween* (measures how separate the clusters are from each other, the more separated the better). For each pair of cluster *i* and *j*, we calculate its average standard deviations and we also calculate the euclidean distance between its centroids. Then, *Fbetween* for clusters *i* and *j* is:

$$Fbetween(i,j) = \frac{EuclideanDistanceFrom\_i\_to\_j}{\sqrt{(AveStdDev_i)^2 + (AveStdDev_j)^2}}$$

Thus, *Fbetween* will be the sum of all pairs of cluster's *Fbetween(i,j)* , over *K*.

-*Fclusters* (measures the number of clusters *K*, "Other things being equal, fewer clusters make the model more understandable and avoid possible over fitting"[13]).

$$Fclusters = 1 - \frac{K - Kmin}{Kmax - Kmin}$$

Kmax and *Kmin* are the maximum and minimum number of clusters.

-*Fcomplexity* (measures the amount of features used to cluster the data, this objective aims at minimizing the number of selected features)

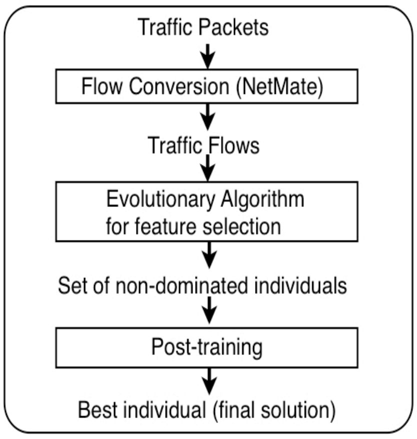
$$Fcomplexity = 1 - \frac{d-1}{D-1}$$

D is the dimensionality of the whole dataset and *d* is the number of employed features.

Instead of combining these objectives into a single objective, this model followed a multi-objective approach, which has the goal to approximate to the Pareto front, or set of non-dominated solutions. Informally, a solution is said to dominate another if it has higher values in at least one of the objective functions (*Fwithin, Fbetween, Fclusters, Fcomplexity*), and is at least as good in all the others. After the objective values for each individual have been assessed, individuals are assigned with ranks, which indicate how many individuals dominate that particular individual. Finally, the fitness of the individuals is inversely proportional to their ranks, which is used to build a roulette wheel that is ultimately used for parental selection under the aforementioned steady state model. The initial population is evolved for 5000 epochs, after which we consider the set of non-dominated individuals from it (individuals whose ranks equal to 1). These individuals correspond to the set of potential solutions. The evolutionary component of the algorithm is then terminated and the best individual in the set of non-dominated solutions (the one that better identifies SSH traffic) is identified in the post-training phase. We take each individual from the set of non-dominated solutions, apply K-Means with its proposed set of features *f* and number of clusters *K*, and label its clusters as SSH or NON-SSH. If the majority of the flows in a cluster have SSH labels, then that cluster is labeled as SSH, otherwise it is labeled as non-SSH. Then, the post-training phase consists of testing each of the non-dominated individual in our training data (used to build the clusters on), to identify the solution with best classification rate, which will be the final solution.

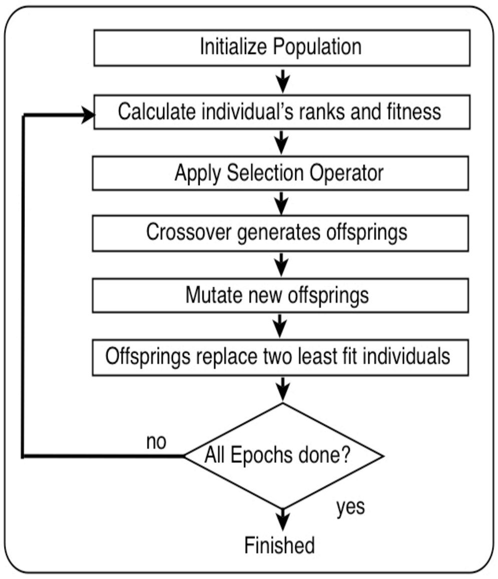
4. Experiments and Results

In traffic classification, two metrics are typically used to quantify performance: Detection Rate (*DR*) and False



Traffic Packets
Flow Conversion (NetMate)
Traffic Flows
Evolutionary Algorithm for feature selection
Set of non-dominated individuals
Post-training
Best individual (final solution)

Figure. 1: System Diagram



Initialize Population
Calculate individual's ranks and fitness
Apply Selection Operator
Crossover generates offsprings
Mutate new offsprings
Offsprings replace two least fit individuals
no
All Epochs done?
yes
Finished

Figure. 2: Evolutionary Component Diagram

Positive Rate (*FPR)*. In this case, *DR* will reflect the number of SSH flows correctly classified, and *FPR* will reflect the number of Non-SSH flows incorrectly classified as SSH. Given that the encrypted traffic only forms a few percent of the traffic in real life, false positive rates are very important on such heavily unbalanced data sets as a performance indicator. As we can observe, a high *DR* and a low *FPR* would be the desired outcomes.

$$DR = 1 - \frac{\#false\_negatives}{total\_number\_of\_SSH\_flows}$$

$$FPR = \frac{\#false\_positives}{total\_number\_of\_non\_SSH\_flows}$$

false_negatives means SSH traffic incorrectly classified as non-SSH traffic.

4.1 K-Means Results

For the basic K-Means algorithm we tried values of *K* from 20 to 400 (table 3). The best combination of *DR* and *FPR* was obtained with *K* = 100, which achieved a *DR* of 98% and a *FPR* of 11%.

Table 3: K-Means Results

Number of clusters	*DR*	*FPR*
20	90%	13%
40	94%	11%
60	97%	11%
80	97%	11%
100	98%	11%
200	98%	15%
300	98%	15%
400	98%	15%

4.2 Semi-supervised K-Means Results

For the semi-supervised model proposed in [9], we first test with the same training data (12250 flows), with *K*= 400 (as done in [9]), but only considering the labels of 5% (613) of the flows when labeling the clusters. The *DR* for that experiment was 90.1% and the *FPR* was 0%. Then we expanded the training data to 32,000 flows, considering the labels of 80, 800, and 8000 of the flows when labeling the clusters (table 4). The highest *DR*, 98.9%, was achieved with 8000 labelled flows. However, the *FPR* was 19%, which is much higher than the other results. Thus, the best combination of *DR* and *FPR* was achieved with only 800 of the flows labelled, which gives a *DR* of 92.0% with a 0% *FPR*. Also, in evaluating the effectiveness of the weighted sampling approach, we obtained a *DR* of 90% and *FPR* of 0%, which is the same we obtained without the weighted sampling method. However, with the weighted sample approach the training data consisted of only 3602 flows, with 180 of them labeled. Thus, from these results we can observe that the weighted sample approach achieves good results with a very small training data.

Table 4: Semi-Supervised Results

K	# labelled flows	*DR*	*FPR*
400	80	90.8%	0%
400	800	92.0%	0%
400	8000	98.9%	19%

4.3 DBSCAN Results

The results obtained with DBSCAN are displayed in figure 3. The y axis represent the detection rate and the x axis represent the *eps*. We experimented with several values for *minPts* (3, 6, 9, 12) and with several values for *eps* (0.01, 0.02, 0.04, 0.06, 0.08). The best results were achieved with *minPts*= 3 and *eps*= 0.02, with a *DR* of 47.4% and *FPR* of 47%.

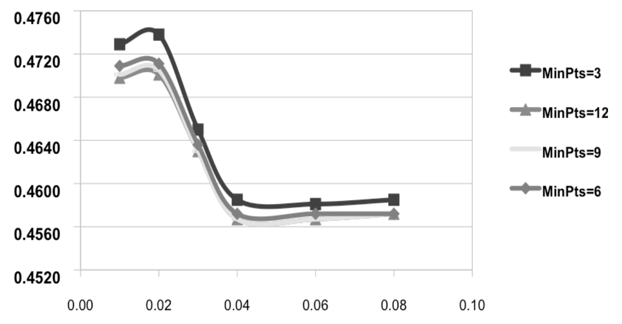


Figure. 3: DBSCAN Results

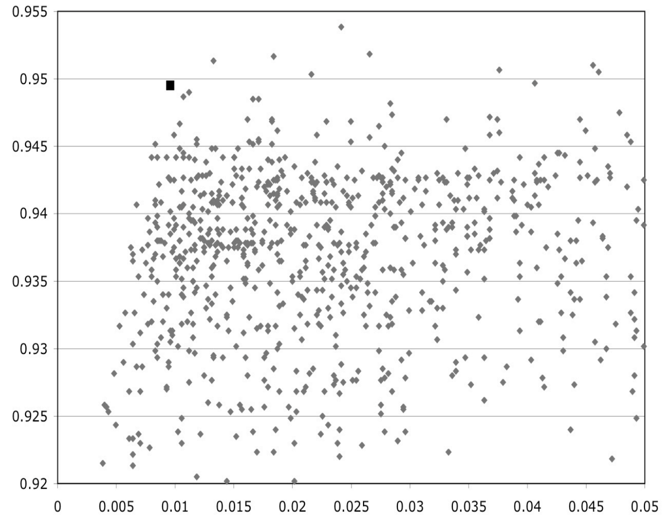


Figure. 4: Non-dominated individuals length 60.

4.4 EM Results

For the EM algorithm we tried with the number of clusters between 100 and 400 (table 5). The best results are achieved with the number of clusters being 400, which gives a high *DR*, 96.4%, while at the same time keeping the *FPR* relatively low, 5%.

Table 5: EM Results

Number of clusters	*DR*	*FPR*
100	0.938	0.079
200	0.958	0.076
300	0.975	0.106
400	0.964	0.05

4.5 MOGA Results

The MOGA was run in two sets of experiments, of 25 runs each. In the first set of experiments the length of the individuals in the population was 60, which allowed the individuals to cluster with a minimum *K* of 2, and a maximum *K* of 23. In the second set of experiments the length of the individuals was 100, allowing the *K* number of clusters to be up to 63. The first set of experiments generated a total of 1173 non-dominated individuals (figure 4). The second

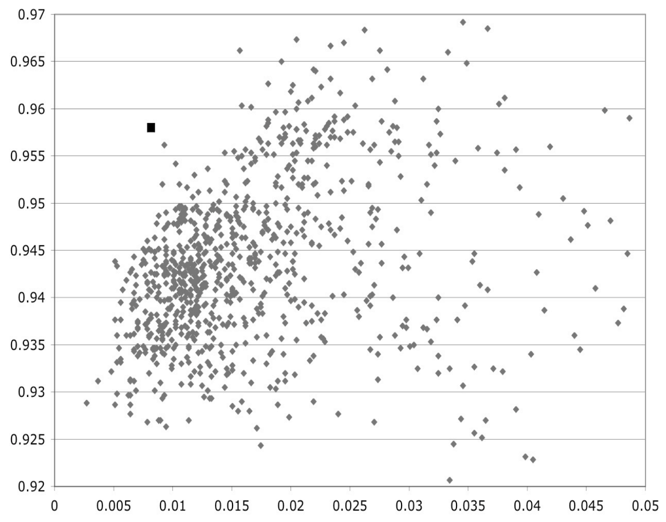


Figure. 5: Non-dominated individuals length 100.

set of experiments generated a total of 869 non-dominated individuals (figure 5). Out of those individuals, we identified in the post-training phase the ones that had the lowest *FPR* (under 1%) and the highest *DR*. We considered those individuals to be our final solutions. Figures 4 and 5 display the plot of the candidate solutions in the post-training phase for both sets of experiments. The x-axis represents the *FPR* and the y-axis represents the *DR*. The best individual in the first set of experiments, which is represented by a larger black square instead of a gray diamond in figure 4, achieved a *DR* of 94.9% and a *FPR* of 0.9% in the post training phase. That same individual achieved a *DR* of 90.9% and a *FPR* of 1.5% in the test data. This final solution employed 22 out of the 38 available features, and clustered the data into 10 clusters. The best individual in the second set of experiments, which is represented by a larger black square instead of a gray diamond in figure 5, achieved a *DR* of 95.8% and a *FPR* of 0.8% in the post training phase. That same individual achieved a *DR* of 93.5% and a *FPR* rate of 0.7% in the test data. This final solution employed 18 out of the 38 available features, and clustered the data into 36 clusters. Thus, these solutions not only achieved very promising results in terms of *DR* and *FPR*, but also considerably decreased the number of employed features, and clustered the data with a relatively low value of K clusters, both very desirable outcomes. Out of the 38 features described in table 1, the 100 bit individual solution (second set of experiments), employed the following 18 features: *proto*, *total_bvolume*, *mean_fpktl*, *max_fpktl*, *std_fpktl*, *std_bpktl*, *min_fiat*, *min_biat*, *mean_biat*, *min_active*, *min_idle*, *sflow_fpackets*, *sflow_fbytes*, *sflow_bpackets*, *sflow_bbytes*, *fpsh_cnt*, *bpsh_cnt*, and *burg_cnt*.

4.6 Time Analysis

One clear advantage of being able to obtain a high *DR* and a low *FPR* with a low number of clusters is the time involved in both, building the models, and testing the data. We measured both the building time, and the test time for all the models here described (table 6). Specifically, we measured the algorithms with the parameters that produced the best results in our previously described experiments. Thus, for the K-Means algorithm we employed *K*= 100, and for the semi-supervised K-Means we employed *K*= 400. It should be noted that in the case of the semi-supervised K-Means the training data was about three times larger than with the other algorithms (32,000 flows). For the DBSCAN algorithm we employed *eps*= 0.02 and *minPts* 3, and for the EM algorithm we employed *K*= 400. For the MOGA, we tested the two individuals that were selected as the best individuals in each of the two conducted sets of experiments. These results show a considerable advantage of the MOGA over all the other algorithms, except DBSCAN. However, the *DR* obtained with DBSCAN was much lower. We conducted these experiments on a standard PC with an Intel(R) Core(TM) 2 Duo CPU T6400@ 2.00 GHz, with 4 GB of memory.

Table 6: Time Results

Algorithm	# cl	**Build time**	**Test time**
K-Means	100	00:02:19	00:49:29
K-Means (semi-sup.)	400	00:22:32	03:07:55
DBSCAN (*mP.=3*)	36	00:04:30	00:05:43
EM	400	00:37:01	03:54:15
MOGA (ind 60)	10	00:00:11	00:18:09
MOGA (ind 100)	36	00:01:55	00:35:10

5. Conclusions

In this work we compared the performance of basic K-Means, semi-supervised K-Means, DBSCAN, EM, and MOGA, to identify encrypted traffic, specifically SSH on our data sets. Our results show that the MOGA obtained better performance in terms of combined *DR*, *FPR*, and computatinoal cost (measured in CPU time). MOGA also clustered the data with a very small value of *K*, which is very desirable for a potential implementation of an encrypted traffic detection system. In this case, MOGA's best individual achieved a detection rate of 93.5% and a false positive rate of 0.7%, whereas it employed only 18 out of the 38 available features and clustered the data in only 36 clusters. The fact that MOGA was able to cluster the data with a much smaller number of clusters, provided a noticeable advantage over the other presented algorithms in terms of the amount of time needed to both building the models and testing/running them on real life data sets. With regards to the semi-supervised model proposed by Erman et. al in [9], we observed that the weighting sampling method could achieve good results with a very small training data. Thus, for future work, we are interested in employing MOGA under a semi-supervised context with a weighted sampling method. Moreover, we also aim to apply this methodology to other types of encrypted traffic such as Skype.

Acknowledgments

This work is supported by ISSNet - NSERC Strategic Network and the CFI new opportunities program. Our thanks to John Sherwood, David Green and Dalhosuie UCIS team for providing us the anonymozied Dalhousie traffic traces. All research was conducted at the Dalhousie Faculty of Computer Science NIMS Laboratory, http://www.cs.dal.ca/projectx.

References

[1] E. Alpaydin. *Introduction to Machine Learning*. MIT Press, 2005.

[2] R. Alshammari and A. Zincir-Heywood. A flow based approach for ssh traffic detection. In *Systems, Man and Cybernetics, 2007. ISIC. IEEE International Conference on*, pages 296–301, Oct. 2007.

[3] R. Alshammari and A. Zincir-Heywood. Investigating two different approaches for encrypted traffic classification. In *Privacy, Security and Trust, 2008. PST '08. Sixth Annual Conference on*, pages 156–166, Oct. 2008.

[4] R. Alshammari and A. Zincir-Heywood. Generalization of signatures for ssh traffic identification. In *IEEE Symposium Series on Computational Intelligence*, 2009.

[5] C. Bacquet, A. Zincir-Heywood, and M. Heywood. An investigation of multi-objective genetic algorithms for encrypted traffic identification. In *International Workshop on Computational Intelligence in Security for Information Systems, CISIS'2009*, 2009.

[6] L. Bernaille, R. Teixeira, I. Akodkenou, A. Soule, and K. Salamatian. Traffic classification on the fly. *SIGCOMM Comput. Commun. Rev.*, 36(2):23–26, 2006.

[7] J. Erman, M. Arlitt, and A. Mahanti. Traffic classification using clustering algorithms. In *MineNet '06: Proceedings of the 2006 SIGCOMM workshop on Mining network data*, pages 281–286, New York, NY, USA, 2006. ACM.

[8] J. Erman, A. Mahanti, and M. Arlitt. Qrp05-4: Internet traffic identification using machine learning. In *Global Telecommunications Conference, 2006. GLOBECOM '06. IEEE*, pages 1–6, 27 2006-Dec. 1 2006.

[9] J. Erman, A. Mahanti, M. Arlitt, I. Cohen, and C. Williamson. Offline/realtime traffic classification using semi-supervised learning. *Perform. Eval.*, 64(9-12):1194–1213, 2007.

[10] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In E. Simoudis, J. Han, and U. M. Fayyad, editors, *Second International Conference on Knowledge Discovery and Data Mining*, pages 226–231. AAAI Press, 1996.

[11] IETF. http://www.ietf.org/.

[12] G. Junior, J. Maia, R. Holanda, and J. de Sousa. P2p traffic identification using cluster analysis. In *Global Information Infrastructure Symposium, 2007. GIIS 2007. First International*, pages 128–133, July 2007.

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

[14] R. Kumar and P. Rockett. Improved sampling of the pareto-front in multiobjective genetic optimizations by steady-state evolution: a pareto converging genetic algorithm. *Evol. Comput.*, 10(3):283–314, 2002.

[15] NetMate. http://www.ip-measurement.org/tools/netmate.

[16] PacketShaper. http://www.packeteer.com/products/packetshaper.

[17] WEKA. http://www.cs.waikato.ac.nz/ml/weka/.

[18] C. V. Wright, F. Monrose, and G. M. Masson. On inferring application protocol behaviors in encrypted network traffic. *J. Mach. Learn. Res.*, 7:2745–2769, 2006.

[19] C. Yang, F. Wang, and B. Huang. Internet traffic classification using dbscan. In *Information Engineering, 2009. ICIE '09. WASE International Conference on*, volume 2, pages 163–166, July 2009.

[20] L. Yingqiu, L. Wei, and L. Yunchun. Network traffic classification using k-means clustering. In *Computer and Computational Sciences, 2007. IMSCCS 2007. Second International Multi-Symposiums on*, pages 360–365, Aug. 2007.

Author Biographies

Carlos Bacquet obtained a Bachelor of Computer Science from Dalhousie University, in Halifax, NS, Canada, in 2008 and is currently a MCS student at Dalhousie University. He also works part time as an R&D consultant for companies based in the Maritime Provinces of Canada. His research interests include but are not limited to the areas of Encrypted Traffic Identification, Genetic Programming and Machine Learning.

A. Nur Zincir-Heywood recieved the Ph.D degree in network information retrieval from the Department of Computer Engineering, Ege University, Izmir, Turkey, in 1998. She is an Associate Professor with the Computer Science Department, Dalhousie University, Halifax, NS, Canada. From 1996 to 1997, she was a Visiting Researcher at the IIMS Research Center, School of Engineering, University of Sussex, Brighton, U.K. Previous to her current position, she was an Assistant Professor with the Department of Computer Engineering, Ege University (1998 - 2000). She has also been involved with Network Technology Workshops of Internet Society as an Instructor from 1997 to 2000. Her research interests include intrusion detection, network security, network management, and network information retrieval. She has published journal and conference papers in these areas, and has been involved in projects concerning network security and information systems. Dr. Zincir-Heywood is a member of the Association of Computing Machinery (ACM) and the Institute of Electrical and Electronic Engineers (IEEE).