Identification over encrypted Channels

Niemczyk, Brandon
brandon.niemczyk@hp.com, HP DVLABS

Rao,Prasad
prasad.rao@hp.com, HP LABS

July 1, 2014

1 Living Document

This paper is a living document, please grab the latest version from
https://raw.githubusercontent.com/bniemczyk/pacumen/master/paper/pacumen.pdf

2 Introduction

The ability to dynamically classify and identify the entities behind network traffic could help us with various network activities. These activities include IP network engineering with network provisioning and network management. This ability is especially important for surveillance for policy compliance and security.

Today's techniques, when applied to traffic that is not hidden rely on the visibility of packet headers, port numbers and the protocol, and stateful reconstruction of sessions. These cues disappear when ssh, SSL or other tunneling mechanisms are used to hide traffic. Whether the hiding is legitimate or not, it is advantageous to identify information about encrypted traffic. Enterprises might proscribe service providers that either allow policy violations or cause a competitive disadvantage. Campus networks could desire to prevent games from clogging up the available bandwidth. From the point of view of a network provider, better classification and understanding of traffic could allow better traffic shaping. Entities interested in policy enforcement could also derive more meta data by such analysis.

We shall use the term *identification problem* to mean the act of identifing traffic behind encrypted channels. There are several published techniques resulting from many years of research in order to classify traffic – most of them rely on machine learning.

We are releasing a simple tool that we call `pacumen`, that relies on simple, easily collectible features in order to solve the *identification problem*. This tool is intended to be be easy to use, and when compared to the academic approaches, need smaller amount of data for training and lesser user intervention for it to function. Additionally, as this

tool is released as an open source project on `github`, we expect users to extend this tool and contribute to its growth. Our main purpose in this exercise to gain insight and understanding to the phenomenon of hiding traffic behind encrypted tunnels and the process of applying machine learning techniques in an attempt to uncover them. This work should both lead to better techniques on both sides of this hide-and-seek game. Lastly, we present some experiments and their results to validate `pacumen`.

3 Related Work/Prior Research

Due to the fundamental nature of the *identification problem* there has been much work over a long period of time that has explored the identification of applications/traffic types over encrypted channels or at least in a payload agnostic manner. Table 1 illustrates a selection of this work.

We cite two attempts at comparing various machine learning algorithms in conjunction with several features to solve the *identification problem*. Lim *et al* [6] compare twenty-two decision tree, nine statistical, and two neural network algorithms on thirty-two datasets with respect to classification accuracy, training time, and (in the case of trees) number of leaf nodes. They measure classification accuracy by mean error rate and mean rank of error rate. They state that most of these algorithms perform similarly with a statistical, spline-based, algorithm called Polyclass working slightly better than other algorithms. Another statistical algorithm, logistic regression, is second with respect to the two accuracy criteria. They find that the most accurate decision tree algorithm is Quest with linear splits, which ranks fourth and fifth, respectively. Although spline-based statistical algorithms tend to have good accuracy, they also require relatively long training times. Polyclass, for example, is third last in terms of median training time. It often requires hours of training compared to seconds for other algorithms. The Quest and logistic regression algorithms are substantially faster. Among decision tree algorithms with univariate splits, C4.5, Ind-Cart, and Quest have the best combinations of error rate and speed. But C4.5 tends to produce trees with comparitively a large number of leaves. Alshamarri *et al* [1] compare AdaBoost, Support Vector Machines, Naive Bayesian, RIPPER and C4.5. They find that C4.5 is the best of these.

Some techniques such as the one by Bernaille *et al*[2] require very few features to be collected – only the first five packets are used. The number five was arrived by trying different number of packets, and different numbers of clusters and finding the pair $\langle 5, 50 \rangle$ to yield the best results.

Karagiannis *et al* [5] enhance their classification mechanism with some social, functional and application level information, and obtain good accuracy.

Roughan *et al* [10] perform classification for the purpose of quality of service(QOS) maintenance.

Authors	ML Technique	Data, Quantity, Features	Accuracy
Williams et al [11]	(i) Baysesian with discretized featured (ii) C4.5 (iii) Naive Bayes (iv) Naive Bayes Tree. Feature reduction using correlation and consistency. Filter vs. wrapper models. Sampling to limit number of flows.	auckland-vi-20010611,2 leipzig-ii-20030221, nzix-ii20000706 4000 Flows per application class ; 22 Features	$> 80\%$
McGregor *et al* [7]	Expectation Maximization(EM)	packet length, inter-arrival time and flow duration.	NA, unsupervised
Zander *et al*[12]	Greedy Forward Feature Search and EM	subset of Table 2	average of 86.5%
Roughan *et al* [10]	Nearest Neighbor (NN) and linear discriminate analysis (LDA) into interactive or transactional	packet, flow, connection, intra-flow, multi-flow	90s
Moore *et al* [9]	Naive Bayes Classifier, Correlation based selection of stronger features	248 flow features – packet length, inter-arrival times	$> 65\%$ per flow. $70 - 97\%$ with Kernel density estimation and and $> 90\%$ with FCBF Filtering
Karagiannis *et al* [5]	graph-based	social, functional, application	$> 95\%$
Bernaille *et al* [2]	K-means clusterting	First 5 packets	$> 80\%$
Lim *et al*[6]	33 algorithms	32 data sets	
Alshammari, Zincir-Heywood [1]	5 algorithms	ssh v.s skype features in Table 2 on old standard data	$> 90\%$
Zhang *et al* [13]	k-NN, Bayesian, random forest	number of packets, bytes transferred, packet size, inter-packet time	variable
Branch *et al*[3]	C4.5	Skype identification. Packet lengths, inter arrival times, large v/s small packets	$> 98\%$

Table 1: Related Machine Learning Approaches Compared

Protocol	Duration of the flow
# Packets in forward direction	# Bytes in forward direction
# Packets in backward direction	# Bytes in backward direction
Min forward inter-arrival time	Min backward inter-arrival time
Std deviation of forward interarrival times	Std deviation of backward interarrival times
Mean forward inter-arrival time	Mean backward inter-arrival time
Max forward inter-arrival time	Max backward inter-arrival time
Min forward packet length	Min backward packet length
Max forward packet length	Max backward packet length
Std deviation of forward packet length	Std deviation of backward packet length
Mean backward packet length	Mean forward packet length

Table 2: Typical feature set for a ML algorithm

The results of Table 1 would perhaps suggest that it is easy to get accuracies of greater than 80% while solving the identification problem using machine learning techniques. While we have had similar successes in attempting to identify Skype against non-Skype, we have not had similar successes with other classification attempts over browser-based applications such as gmail.

Absorbing the work that went behind all these papers gave us much better insight and intuition into the nature of the identification problem and the techniques to solve it than if we were to attempt solving it from scratch.

4 A quick introduction to Machine Learning

Given that this problem has been addressed using machine learning techniques, it is fruitful to introduce these properly (beyond presenting a laundry list of machine learning algorithms). In his book [8], Tom Mitchell defines *learning* as: A computer program is said to **learn** from experience E with respect to some tasks T and some performance measure P if its performance at tasks in T improves with E.

When using machine learning for the act of assigning entities to classes, the result of learning is a *classifier* which can be applied to instances not seen during training.

Our main goal, while solving the identification problem is not to debate the meaning of the word *learning* but to find algorithms and techniques that satisfy the above definition and to devise some ourselves.

The act of designing a machine learning system incudes firstly the design or choice of the training experience. Secondly, we must choose or design a target function that helps us with the act of classification. Thirdly, we need to choose a representation of this target function. We need to arrive at approximation schemes to evaluate this target function

for values not in the training set. We will describe how we performed all of these steps listed above in the next section.

4.1 Issues while designing machine learning systems

The list of Machine Learning algorithms continues to grow and some of them are being used heavily in the real world. Understanding the appropriateness of these algorithms to the problem being solved, and following up this understanding with a decision of which algorithm to use is difficult. Also, the decision of having to design a new algorithm is not an easy one, and needs justification – often by appealing to the special features of the problem at hand.

It is often asserted in literature that a large amount of data with simple algorithms outperforms sophisticated algorithms with smaller amounts of data. The central question here is: *How much training data is sufficient?* How can we quantify the confidence in the learning done on the training data?

How much can we use prior knowledge in these systems. For instance is it OK to pre-process data – by ignoring some particular features, even if the knowledge guiding this decision is approximate?

When should re-training be done? Should we do the same kind of training as we did before, or should we modify the training – say, by adding/deleting features?

What form should the classifier take? Should we let the learning system decide this?

4.2 Performance Issues

Sophisticated machine learning algorithms allow parameterization of the extent to which they will fit the model to the training data. It is easy to fall into the trap of overfitting the data – in which case results in the laboratory look very good. These models do not often perform as well on real world data.

In our case, the application of standard machine learning algorithm such as *BayesianLogisticRegression* to the problem of distinguishing Skype from everything else had a *precision* of 1 which means that it had no false positives and a *recall* of 0.872. We are in the process of determining the extent of overfitting in this and other good results by applying the same technique to 10x the amount of data.

5 Our Approach

Our approach is to break the stream up by time buckets and try various weak classifiers on buckets and then adjust our confidence over time. The reason for using time based

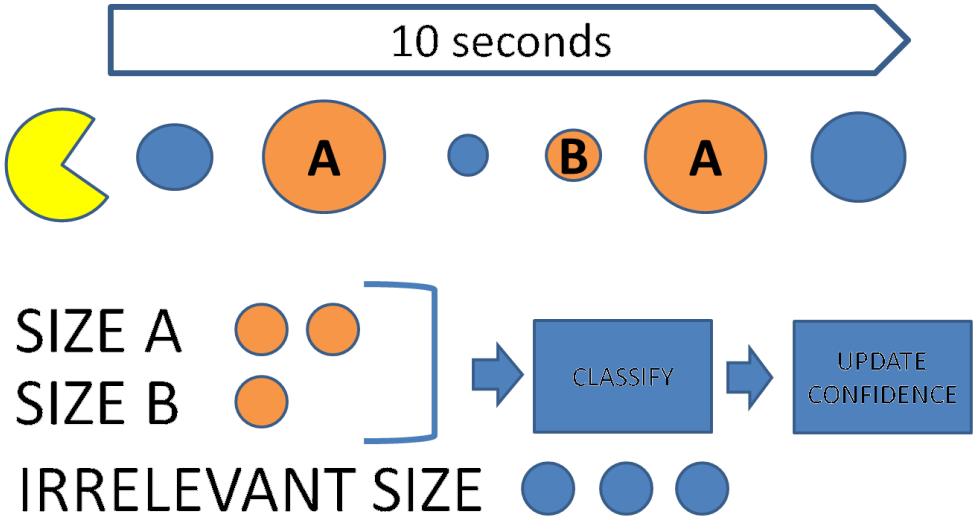

Figure 1: Illustration of bucketing sizes for a time window

windows instead of a set amount of packets is that intuitively there is a better chance of identifying the application even if it is mixed with traffic from other applications on the tunnel. See 5.1 for classifying each time window and 5.2 for information on updating the confidence over time.

We looked at two primary *features* - packet size counts and timing between packets. In our experience the timing was too noisy for accurate classification, but the packet size counts and relations between them are very useful. And we can implicitly include some timing information by generating our feature vectors as a function of time. So what we do is generate a single 65536 dimension feature vector where each dimension is a count of the number of packets with a size that is the same as the "number" of that dimension.

As an illustration, if you take say 100 packets and treat that as your feature vector, and the user is 1) using skype and 2) watching a youtube video. It is likely you will get no traffic for skype or at least not get consistent numbers of packets in that 100 length sequence due to being overwhelmed with video data. On the other hand in a 10 second window you can expect a certain amount of traffic from any particular application you are looking to identify, and our feature selection will ideally discard most of the noise.

Figure 1 displays a simple workflow with a 10 second window that saw 6 packets, 3 of which were sizes that the classifier deems relevant.

5.1 Algorithms used to classify 10 second windows of traffic

In order to be able to easily update our confidence using section 5.2 our window classifier needs to return an estimate of the probability of the feature vector for both our negative and positive classes. We should emphasize that it is not a probability that it belongs to the positive class.

Likelihood function Assume traffic is generated by a mixed gaussian and estimate the density function using the multi-variate normal likelihood function (Equation (5.1)).

Decision trees See section 5.1.1. This performs significantly better than the Likelihood function.

$$L(\mathbf{x}) = e^{-\frac{1}{2}\ln(|\Sigma|)-\frac{1}{2}(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)-\frac{k}{2}\ln(2\pi)} \tag{5.1}$$

where

$\mathbf{x}$ = column feature vector

μ = Sample mean column vector

Σ = Sample covariance matrix

$|\Sigma|$ = determinant of Σ

k = number of dimensions (65536)

5.1.1 Decision Tree Algorithm

See Figure 2 for an example of a Decision Tree that we would like to create.

Algorithm 5.1 describes the decision tree creation algorithm in pseudocode. These trees give much better accuracy than the likelihood function and in fact get perfect accuracy with Skype, which most of the existing work has used. The bias parameter is important to help avoid overfitting, and can be set by the user. Additionally, the tools provided can search for the best bias.

5.2 Using Bayes Rule to update our confidence

For each network stream we maintain a probability that the stream contains data from our positive class ($P(CLASS|DATA)$). For every 10 seconds of data we then create a feature vector and estimate $P(DATA|CLASS)$ for both the positive and negative classes using either the likelihood function or decision trees. Once we have this estimate we can update our stored prior (confidence if you prefer) using Bayes' Rule (Equation (5.2)).

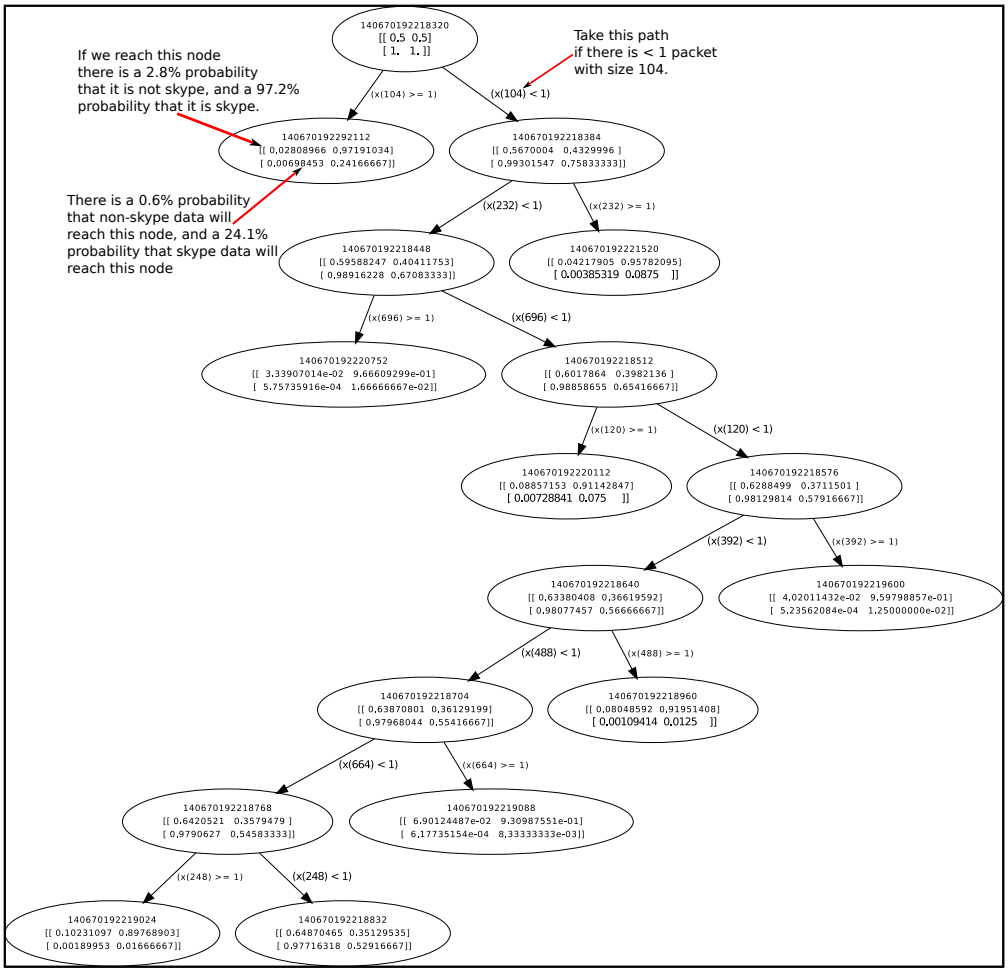

Figure 2: Example Skype/SSH decision tree

Algorithm 5.1 Decision Tree Creation Algorithm

function SELECTFEATUREANDCUTOFF($data, bias$)
  $col \leftarrow \perp$
  $cut \leftarrow \perp$
  $max \leftarrow -\infty$
  for all $\{column, thresh | column, thresh \in features \times values\}$ **do**
    $igain \leftarrow$ information gain ratio using column and thresh

▷ We use the *bias* parameter to bias our selection to column/thresh pairs that look for a decrease in the probability of the positive class on the $<$ side, this is so that we do not learn that a protocol "will not have these sizes"

    $score \leftarrow (P(col >= thresh|positive) * \frac{P(col>=thresh|positive)}{P(col>=thresh|negative)} * igain^{bias})^{\frac{1}{2+bias}}$
    if $score > max$ **then**
      $max \leftarrow score$
      $col \leftarrow column$
      $cut \leftarrow thresh$
    end if
  end for
  return col, cut
end function

function EXPANDNODE($data, bias$)
  $p \leftarrow$ probability of positive class given $data$
  if $p < \epsilon \vee p > 1 - \epsilon$ **then**
    return
  end if
  $f, c \leftarrow$ SelectFeatureAndCutoff($data, bias$)
  $low \leftarrow \{x | x \in data \wedge x[f] < c\}$
  $high \leftarrow \{x | x \in data \wedge x[f] >= c\}$
  ExpandNode(low)
  ExpandNode($high$)
end function

ExpandNode(training data)

$$P(CLASS|DATA) = \frac{P(DATA|CLASS)P(CLASS)}{\Sigma_{C \in CLASSES} P(DATA|C)P(C)} \tag{5.2}$$

6 Experiments and Results

We collected the data using the `--nflog` option in iptables, for outbound packets only. We extracted the features using the python wrapper to libcap –`scapy` in linux and `winpcapy` in windows.

The pacumen tool can be run directly within a `iPython` notebook and setting a variable to point to the directory with the pcap files. In order to run algorithms in Weka[4] we provide a converter.

7 Discussion

We resort to standard measures of precision, recall and f-measure to demonstrate the efficacy of our approach.

$$precision = \frac{Number\ of\ Correctly\ classified\ instances}{Total\ number\ of\ positive\ classifications}$$

$$recall = \frac{Number\ of\ Correctly\ classified\ instances}{Total\ number\ of\ class\ members}$$

$$f - measure = 2.\frac{precision.recall}{precision + recall}$$

We ran our experiments on our machine learning algorithms and also on standard machine learning algorithms in Weka. Table 3 shows the result.

The results are comparitively worse for the classes `firefox.gmail` and `firefox.linkedin`. The extra rows `gmail` and `linkedin` were created by combining the firefox and chrome versions of these classes. It can be observed that this union yielded better results, leading to the conclusion that the firefox and the chrome versions of the classes were often mistaken for one another.

References

[1] Riyad Alshammari and A Nur Zincir-Heywood. Machine learning based encrypted traffic classification: identifying ssh and skype. In *Computational Intelligence for*

	J48			LADTree			BLR			Our Tree		
	P	R	F	P	R	F	P	R	F	P	R	F
chrome.facebook	.92	.79	.85	.87	.82	.84	.85	.70	.77	.70	.96	.81
chrome.gmail	.64	.32	.43	.60	.38	.46	.59	.54	.56	.38	1.0	.55
chrome.linkedin	.51	.56	.54	.63	.40	.49	.78	.72	.75	.95	.43	.59
firefox.facebook	.84	.64	.72	.86	.73	.79	.88	.67	.76	.33	1.0	.50
firefox.gmail	.30	.15	.21	.68	.34	.46	.88	.67	.76	.18	1.0	.31
firefox.linkedin	.30	.15	.21	.67	.31	.42	.82	.35	.49	.50	1.0	.66
linkedin	.71	.45	.64	.67	.47	.55	.69	.57	.62	–	–	–
gmail	.86	.50	.78	.76	.72	.74	.80	.78	.79	–	–	–
skype	.89	.90	.90	.95	.90	.92	1.0	.87	.93	1.0	1.0	1.0

Table 3: Results of running some standard machine learning algorithms in Weka upon our data. The columns P,R,F stand for precision, recall and f-measure respectively. BLR stands for Bayesian Logistic Regression.

Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium on, pages 1–8. IEEE, 2009.

[2] Laurent Bernaille, Renata Teixeira, Ismael Akodkenou, Augustin Soule, and Kave Salamatian. Traffic classification on the fly. *ACM SIGCOMM Computer Communication Review*, 36(2):23–26, 2006.

[3] Philip A Branch, Amiel Heyde, and Grenville J Armitage. Rapid identification of skype traffic flows. In *Proceedings of the 18th international workshop on Network and operating systems support for digital audio and video*, pages 91–96. ACM, 2009.

[4] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[5] Thomas Karagiannis, Konstantina Papagiannaki, and Michalis Faloutsos. Blinc: multilevel traffic classification in the dark. In *ACM SIGCOMM Computer Communication Review*, volume 35, pages 229–240. ACM, 2005.

[6] Tjen-Sien Lim, Wei-Yin Loh, and Yu-Shan Shih. A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms. *Machine learning*, 40(3):203–228, 2000.

[7] Anthony McGregor, Mark Hall, Perry Lorier, and James Brunskill. Flow clustering using machine learning techniques. In *Passive and Active Network Measurement*, pages 205–214. Springer, 2004.

[8] Tom M Mitchell. Machine learning. wcb, 1997.

[9] Andrew W Moore and Denis Zuev. Internet traffic classification using bayesian analysis techniques. In *ACM SIGMETRICS Performance Evaluation Review*, volume 33, pages 50–60. ACM, 2005.

[10] Matthew Roughan, Subhabrata Sen, Oliver Spatscheck, and Nick Duffield. Class-of-service mapping for qos: a statistical signature-based approach to ip traffic classification. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 135–148. ACM, 2004.

[11] Nigel Williams, Sebastian Zander, and Grenville Armitage. A preliminary performance comparison of five machine learning algorithms for practical ip traffic flow classification. *ACM SIGCOMM Computer Communication Review*, 36(5):5–16, 2006.

[12] Sebastian Zander, Thuy Nguyen, and Grenville Armitage. Automated traffic classification and application identification using machine learning. In *Local Computer Networks, 2005. 30th Anniversary. The IEEE Conference on*, pages 250–257. IEEE, 2005.

[13] Jun Zhang, Chao Chen, Yang Xiang, and Wanlei Zhou. Robust network traffic identification with unknown applications. In *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*, pages 405–414. ACM, 2013.