Real Time Identification of SSH Encrypted Application Flows by Using Cluster Analysis Techniques

Gianluca Maiolini[1], Andrea Baiocchi[2], Alfonso Iacovazzi[2], and Antonello Rizzi[2]

[1] Elsag Datamat, Automation, security and transportation division,
V. Laurentina 760, 00143 Rome, Italy
gianluca.maiolini@elsagdatamat.com
[2] INFOCOM Dept., University of Roma "Sapienza"
Via Eudossiana 18 - 00184 Rome, Italy
{name.surname}@uniroma1.it

Abstract. The identification of application flows is a critical task in order to manage bandwidth requirements of different kind of services (i.e. VOIP, Video, ERP). As network security functions spread, an increasing amount of traffic is natively encrypted due to privacy issues (e.g. VPN). This makes ineffective current traffic classification systems based on ports and payload inspection, e.g. even powerful Deep Packet Inspection is useless to classify application flow carried inside SSH sessions. We have developed a real time traffic classification method based on cluster analysis to identify SSH flows from statistical behavior of IP traffic parameters, such as length, arrival times and direction of packets. In this paper we describe our approach and relevant obtained results. We achieve detection rate up to 99.5 % in classifying SSH flows and accuracy up to 99.88 % for application flows carried within those flows, such as SCP, SFTP and HTTP over SSH.

Keywords: Traffic analysis, statistical traffic classification, SSH, cluster analysis, k-means.

1 Introduction

The control of QoS becomes a very important stake, even more with the arrival of new applications with very different profiles, such as real time multimedia. These applications have very different behavior according to packet sizes, offered traffic profile, transport level protocol (UDP/TCP) and different requirements on throughput, transit delay, jitter, packet loss rate.

Traffic shaping and scheduling techniques for edge routers aim at managing bandwidth resources according to QoS policy model. Before applying QoS policy it is important to classify correctly application flows. Most routers use port based classification of protocols but new applications are forwarded using well-known ports, i.e 80. Deep packet inspection (DPI) systems can be quite effective in recognizing applications, thanks to IP payload analysis, even if they fail in classifying ciphered flows. From provider side, it will be very important to recognize applications encoded within ciphered flows in order to apply QoS policies; in fact encoded traffic could contain

L. Fratta et al. (Eds.): NETWORKING 2009, LNCS 5550, pp. 182–194, 2009.

real time applications, transactional applications (ERP, finance,..), comfort application (web, mail,..) and so on. In this situation, future requirements could consist in optimizing performance in terms of QoS for application natively encoded.

Secure Shell or SSH is a network protocol that allows data to be exchanged using a secure channel between two networked devices. It is often used to login to a remote computer but it is also applied for tunneling, file transfer and forwarding arbitrary TCP ports over a secure channel between a local and a remote computer. What makes the detection of this protocol interesting is that its traffic is encrypted. Thus any payload analysis based classification method is irrelevant since the payload is encrypted. Actually DPI technology cannot recognize application delivered within SSH flows.

The objective of our work is to develop a real time system to recognize and classify SSH flows by analyzing statistical features of first IP packets belonging to a SSH connection, such as arrival times, directions and lengths. By recognition we mean identifying which flows belong to SSH protocol as opposed to other application level protocols. By classification we mean to identify the kind of service carried within each SSH connection, such as SCP, SFTP and HTTP over SSH. Experiments show that our approach permits us to achieve great recognition accuracy up to 99.2% for SSH identification and, once SSH has been identified, applications in those SSH tunnels are classified with accuracy up to 99.8%.

The paper is organized as follows. Section II establishes the position of this work, relative to earlier research. Problem statement has been discussed in section III. In section IV we have described dataset creation, in particular data collection and preprocessing. The machine learned based approaches for real time SSH classification are presented in section V. Experimental results are presented in section VI, and conclusions are drawn in section VII.

2 Related Works

Different approaches to traffic classification have been developed, using information available at IP layer such as inter-arrival times, bytes transferred, packet size. Some proposals [4][5] need also semantically complete TCP flows as input.

In [1], Karagiannis et al. developed a heuristic that uses social, functional and application level behaviours of a host to identify all traffic flows originating from it. This approach, although really innovative, is tailored onto a specific source host.

Salgarelli et al. [2] used only size and inter-arrival time of first *n* packets to create a statistical descriptor (a Fingerprint) of an application layer protocol: this fingerprint is then used to measure the similarity of a certain flow to the corresponding protocol.

The Hidden Markov Models (HMM) theory is used in [3]: packets size and inter-arrival time are used to build a model describing a certain protocol. The results of the training phase is a HMM model describing the behaviour of each protocol. Even though this approach can classify distinct encrypted applications, its performance on SSH is (76% detection rate and 8% false negative) is not as good as well known application traffic such as WWW and instant messaging.

Moore et al. [4] used a supervised machine learning algorithm called Naive Bayes (and its generalization, Kernel Estimation) on a wide set of characteristics (tens or hundreds), as flow duration, packets inter-arrival time and payload size and their

statistics (mean, variance...). Moreover, they use a filtering technique to identify the best characteristics to be used with the mentioned methods.

A number of works [5][6][7] rely on unsupervised learning techniques. McGregor et al. [5] explore the possibility to use cluster analysis to group flows using transport layer attributes, but they do not evaluate the accuracy of the classification. Zander et al. [6] extend this work using another Expectation Maximization (EM) algorithm named Autoclass. They also analyze the best set of attributes to use. Both these works only test Bayesian clustering technique trained by an EM algorithm, which has a slow learning time.

Bernaille et al. [7] use faster clustering algorithms representing data in different spaces: K-means and Gaussian Mixture Models (GMM) for euclidean space and Spectral clustering in HMM based space. The only features they use are packet size and packet direction: they demonstrate the effectiveness of these algorithms even using a small number of packets (e.g. the first four of a TCP connection).

Alshammari et Al [8], work attempted to classify/identify applications services running over SSH. They have shown the utility of two supervised learning algorithms AdaBoost and RIPPER for classifying SSH traffic without using features such as payload, IP addresses and source/destination ports. Results indicate that a detection rate of 99% and a false positive rate of 0.7% can be achieved using RIPPER. Moreover, promising preliminary results were obtained when RIPPER was employed to identify which service was running over SSH. They can recognize applications inside SSH flows such SCP and SFTP with accuracy up to 99.8% but they have performed off-line analysis on complete traces. We aim at classifying applications inside SSH flows in real time mode just analyzing the firsts 4 packets after SSH negotiation. We rely on K-means cluster analysis machines algorithm.

3 Problem Statement

In this paper, we focus on the classification of IP flows generated from network applications communicating through TCP protocols. Our objective is to recognize SSH flows out of other applications such as HTTP, FTP, POP3, etc. and, once that is accomplished, to identify which service is actually carried within the encrypted SSH tunnel. Then, we first need to define exactly what we mean for TCP flow.

Definition: A flow F is the bi-directional, ordered sequence of IP packets exchanged during a TCP connection.

Within a TCP connection, application level data are delivered as well as control packets, such as those related to three way-handshake (RFC-793) and TCP ACK packets. So, TCP flow will be composed by packets from SYN (PK_0) to FIN (PK_{N-1}). Each flow could be seen as a sequence of (PK_0, …., PK_{N-1}), where PK_j represents the j-th IP packet exchanged during TCP connection. Since we aim at classifying application flows relying on statistical features of IP packets, such as length, direction and absolute-arrival times, we will characterize each TCP flow F as an ordered sequence of N-tuples (d_j, l_j. t_j), with $0 \leq j \leq N-1$, where:

- $d_j \in [0,1]$ where 1 encodes the direction detected for SYN packet and 0 the opposite direction;
- l_j length of IP PK_j in bytes;

- $t_j = T_j - T_0$, where T_j is the timestamp of PK_j at capture point and T_0 is the timestamp of the PK_0 at the capture point.

The packet length ranges between a minimum and a maximum. The latter is the MTU (Maximum Transmission Unit) of the interfaces crossed by TCP connections packets. In all experiments we found out MTU=1500 bytes has never been exceeded, which is just the largest allowed MTU of most Ethernet LANs and hence most of the Internet [10]. As for the minimum length, it corresponds to those carrying a TCP ACK and is denoted as l_{ACK} in the following. It is the smallest length detectable for a TCP packet as we tested during our experiments and as RFC 793 refers, typical values ranging between 40 and 56 bytes, depending on options in the TCP and IP headers.

4 Data Set Creation

Given our aim as stated in the introduction, we assume a trained machine learning approach, exploiting cluster analysis. To that end, we need both a test and a train data set. A data set for our purposes is composed of a collection of flows in the sense defined in Section III along with metadata per flow, reporting the *known* application layer protocol the flow belongs to, the absolute timestamp of its first packet (T_0), the capture date and location.

Knowing the application protocol each flow belongs to is needed to reliably train our algorithm. Since publicly available traces have payloads stripped off (for obvious privacy reason, e.g. CAIDA traces) and classification results cannot be checked reliably, we resorted to artificial traffic carefully generated by exploiting network premises at the University campus, the Elsag Datamat site and a private home. This way we encompass three major kinds of Internet access points: institutional, business and domestic. The controlled traffic generation is a must specifically for collecting SSH traces whose service content is known, i.e. to further label each SSH flow with a metadata reading which service it is carrying among SCP, SFTP and HTTP.

4.1 Data Collection

IP traces generation of TCP based application: We focus our attention on four different plain application layer protocols, namely HTTP, FTP-control, POP3 and of course SSH, which seem to be the ones accounting for the majority of traffic flows in the INTERNET (except of peer-to peer traffic). As for HTTP and FTP-Control (FTP-C in the following), we collected traffic traces coming from the Networking Lab at our Department. By means of automated tools mounted on machines within the Lab, thousands of web pages have been downloaded in a random order, over thousands of web sites distributed in various geographical areas (Italy, Europe, North America, Asia). FTP sites have been addressed as well and control FTP session established with thousands remote servers, again distributed in a wide area. The generated traffic has been captured on our LAN switch configuring a mirroring port; we verified that the TCP connections bottleneck was never the link connecting our LAN to the big Internet. This experimental set up, while allowing the capture of artificial traffic that (realistically) emulates user activity, gives us traces with reliable application layer protocol classification. In order to complete our traces repository, in the next section we describe how we collected SSH traces.

SSH IP traces generation: Our data collection approach is to simulate possible network scenarios using one or more computers to capture the resulting traffic. In order to have realistic traces and technology independent implementations of SSH (version 2) protocol, we used computers with heterogeneous operative systems, namely Linux and Windows. We simulate SSH connections by connecting three client computers deployed in three different LAN to one server. As shown in figure 1, client LANs and SSH server have been connected to the Internet by using different geographic links. We run the following SSH services: SCP, SFTP and HTTP over SSH. SCP and SFTP are transfer file services natively available on OpenSSH [9]. In particular we downloaded/uploaded files from clients to server using both SCP and SFTP protocols collecting eight thousands flows. HTTP over SSH traces have been collected downloading web pages through SSH tunnels (one SSH tunnel for each HTTP session). We get four thousands of flows.

SSH connections can tunnel several TCP flows at the same time: we are working in the case where each flow is assigned by SSH a separated channel, each with specific SSH identifier. Finally we will consider flows without SSH compression feature.

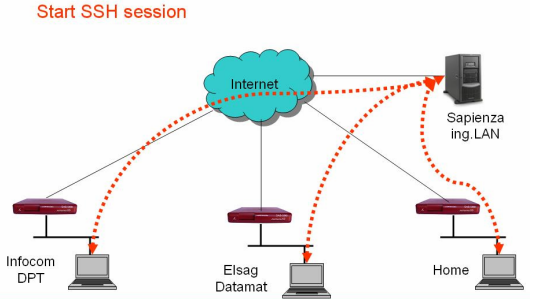

Fig. 1. Platform used to generate SSH traffic: SSH server is inside the University campus network; clients are at University, Elsag Datamat and a private home premise, respectively

4.2 Data Set Creation: Pre-processing of Traces

In order to create data sets we pre-processed collected traffic traces. In particular we think that removing packets related to TCP control messages from each flow F can help us highlighting the differences among various applications. Therefore we remove from each flow F packets related to:

- Three-way handshake of TCP: $PK_{0=SYN}$, $PK_{1=SYN-ACK}$, $PK_{2=ACK}$;
- TCP ACK packets, i.e. those packets carrying only a TCP level ACK and no payload data;
- Retransmitted packets.

According to TCP protocol (RFC 793) the third packet ($PK_{2=ACK}$) of each TCP connection flow F carries an ACK. In order to remove ACK packets and TCP header length at the same time, we detect $PK_{ACK} = <d_{ACK}, l_{ACK}, t_{ACK}>$ of each session, where:

- d_{ACK} is 1, because ACK direction in three way handshake is always consistent with that of SYN packet;
- l_{ACK}, is the length of packet containing TCP ACK;

- t_{ACK}, is the relative capture time of that packet.

So, for each flow n packets will be pre-processed as:

$$\underline{PK_j}^* = <\delta_j = d_j, \lambda_j = l_j - l_{ACK}, \tau_j = t_j - t_{ACK}>, \quad j \in [3,..., N - N_{ACK} - 1]$$

where N_{ACK} is the number of ACK packets detected in the pre-processed flow. Packets with $\lambda_j=0$ are removed and packets shown in $\underline{PK_j}^*$ will contain bytes concerning just application contribution. In particular, $\lambda_j \in [0, l_{MTU} - l_{ACK}]$.

In order to make our analysis in real time, we need to run our classification method exploiting the very firsts packets of each flow. After tests and analysis of results we set how many packets will be required to strike a convenient trade-off between high classification accuracy and an acceptable classification delay. So our dataset will be composed as shown in Figure 2. In our dataset we collected traces belonging to the following application layer protocols: HTTP, FTP-C, POP3 and SSH.

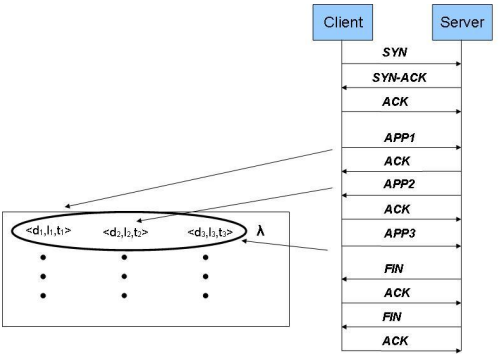

Fig. 2. Pre-processing TCP flows

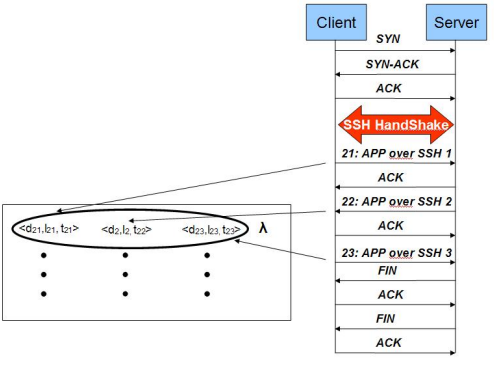

Fig. 3. Pre-processing of SSH flows

Once SSH application flows has been detected, we aim at identifying application within SSH tunnels. Then, we further process SSH flows by removing packets related

to the SSH initial handshake (see Figure 3). We consider the following services inside encrypted SSH tunnels: SCP, SFTP and HTTP over SSH.

5 Classification Method

5.1 A K-Means Based Approach

In this section some details about the adopted classification system are exploited. Basically a classification problem can be defined as follows. Let $P : X \rightarrow L$ be an unknown oriented process to be modeled, where X is the domain set and the codomain L is a label set, i.e. a set in which it is not possible (or misleading) to define an ordering function and hence any dissimilarity measure between its elements.

If P is a single value function, we will call it *classification function*. Let S_{tr} and S_{ts} be two sets of input-output pairs, namely the training set and the test set. We will call *instance* of a classification problem a given pair (S_{tr} , S_{ts}) with the constrain $S_{tr} \cap S_{ts} = \emptyset$. A classification system is a pair (M , TA_i), where TA is the training algorithm, i.e. the set of instructions responsible for generating, exclusively on the basis of S_{tr}, a particular instance $\overline{\mathrm{M}}$ of the classification model family M, such that the classification error of $\overline{\mathrm{M}}$ computed on S_{ts} will be minimized. The *generalization capability,* i.e. the capability to correctly classify any pattern belonging to the input space of the oriented process domain to be modeled, is for sure the most important desired feature of a classification system. From this point of view, the mean classification error on S_{ts} can be considered as an estimate of the expected behavior of the classifier over all the possible inputs. In the following, we describe a classification system trained by an unsupervised (clustering) procedure.

When dealing with patterns belonging to the R^n vectorial space we can adopt a distance measure, such as the Euclidean distance; moreover, in this case we can define the prototype of the cluster as the centroid (the mean vector) of all the patterns in the cluster, thanks to the algebraic structure defined in R^n. Consequently, the distance between a given pattern x_i and a cluster C_k can be easily defined as the Euclidean distance $d(x_i ; \mu_k)$ where μ_k is the centroid of the pattern belonging to C_k:

$$\mu_k = \frac{1}{\mu_k} \sum_{x_i \in C_k} x_i$$

A direct way to synthesize a classification model on the basis of a training set S_{tr} consists in partitioning the patterns in the input space (discarding the class label information) by a clustering algorithm (in our case, by the K-means).

Successively, each cluster is labeled by the most frequent class among its patterns. Thus, a classification model is a set of labeled clusters (centroids); note that more than one cluster can be associated with the same label, i.e. a class can be represented by more than one cluster. Assuming to represent a floating point number with four bytes, the amount of memory needed to store a classification model is $K \cdot (4 \cdot n + 1)$ bytes, where n is the input space dimension and assuming to code class labels with one byte. An unlabeled pattern x is classified by determining the closest centroid μ_{i} (and thus the closest cluster C_{i}) and by labeling x with the same class label associated with C_i.

It is important to underline that, since the initialization step of the K-Means is not deterministic, in order to compute a precise estimation of the performance of the classification model on the test set S_{ts}, the whole algorithm must be run several times, averaging the classification errors on S_{ts} yielded by the different classification models obtained in each run.

Normalization

This paragraph describes how data sets have been normalized. The available data is split in training set and test set. The training and test data sets are normalized in order to guarantee that each feature will contribute to distance computations with equal weights; for each feature we adopted the "affine normalization", consisting in:

$$y = \frac{x - \min}{\max - \min}$$

where x is the value we want to normalize, *max/[min]* is the maximum/[minimum] value of x in a reasonable range of possible values of x and y is the normalized value.

The direction takes the values in the domain (0,1), and doesn't need to be normalized. As concerns the lengths, min and max values are respectively 0 byte and l_{MTU}– l_{ACK} bytes. Absolute arrival time was intended between 0 and 200 sec. During normalization, timestamps over two hundreds seconds has been normalized to 1.

Cross-validation

In order to choose the optimal number of clusters to be used in the K-means clustering procedure, we have performed cross validation. It help us to estimate "a priori" the performance of the classification system using only the training set and give us an estimate of the homogeneity of training set. The cross-validation is a function which receives as inputs the training set (containing *a x N* patterns, in which *a* is the number of applications and *N* is the number of patterns for each application), the length range of clusters' number to use and the number of parts to split the training set (*n*-fold cross-validation), and it returns a vector of length *l* containing the estimated accuracy for each classifier synthesized during the cross-validation procedure. In more detail, the function divides in *n* parts (simply called sub-datasets) the train dataset, assigning *n* patterns for each application in a random way to each part. All the new created datasets will have he same number of patterns for each application and will be balanced in the same way of the original dataset. Next, for each value of *k* belonging to a considered range of reasonable values, the function calculates the classifier performances taking in turn as test set one of the n sub-dataset and as training set the remaining *n-1*. Therefore the training procedure will be run *n* times, as long as each of the sub-dataset *n* has been test set once. The results are inserted into a matrix $\prod$ of *k x n* size:

$$\Pi = \begin{pmatrix} p_{1,1} & p_{1,2} & .. & p_{1,n} \\ p_{2,1} & .. & .. & .. \\ .. & .. & .. & .. \\ p_{k,1} & p_{1,2} & .. & p_{k,n} \end{pmatrix}$$

where the $p_{i,j}$ represents the mean accuracy of the classifier that has been obtained by the i^{th} dataset as test set and by clustering data fixing k equal to j. The mean accuracy is the average of the accuracies on the single application $acc_h^{(i)}(k)$ divided by the application number.

$$p_{k,i} = \frac{1}{a}\sum_{h=1}^{a} acc_h^{(i)}(k)$$

Finally the function computes a further average of the accuracies in the matrix $\prod$:

$$\pi(k) = \frac{1}{n}\sum_{i=1}^{n} p_{k,i}$$

obtaining in this way the $\pi(k)$ performance vector of the classifier, as a function of the number of clusters fixed in the clustering procedure. Finally, the optimal number of clusters to be used in the classification model is determined as follows:

$$k = \arg\max_{1 \le k \le l} \{\pi(k)\}$$

Actually, in this way it is very likely that the highest k value in the range will be chosen because the performances of the classifier tends to improve by increasing of the number of clusters. However, the cross-validation purpose is to find the number k of clusters representing a trade-off between performances and complexity of clustering algorithm. We have chosen the minimal value of k after which the rate of the performance indicated from vector π increases slower (or decreases).

5.2 On-Line Procedure for Capturing and Classifying Application Flows

In this section, we will describe how our algorithm works. Our method is divided into four phases.

First phase: traffic retrieving. This phase consists in connecting a network protocol analyzer, such Wireshark, to the network we want to investigate. In the case of LAN traffic classification, Wireshark workstation will be connected to a mirroring port of the edge switch in order to sniff all traffic traces flowing through the geographical link connecting LAN to the Internet.

Second phase: detecting TCP flows. Wireshark allows our tool to manage traffic in real time. As SYN packet is detected, related IP source and destination are identified as peers involved in a new TCP flow. Datagram exchanged by those IP addresses are retrieved and stored until the number of packets required for classifying application has been reached. Number of packets analyzed for each flow is settable (for instance: six after three-way handshake). This method enables us identifying TCP flows through IP addresses and TCP flags, as well as retrieving plain information for classifying applications such as lengths, direction and arrival time of each packet.

Third phase: preprocessing and normalization phase. Each TCP flow previously detected includes TCP control information as well as retransmitted packets, so lists of TCP flows are preprocessed according to specifications described in Section 4.2 and normalized as depicted in Section 5.1, in “normalization” sub section.

Fourth phase: classification. Output of the previous phase is a list of application flows processed and normalized. Our k-means based classification machine is

configured off-line through process shown in Figure 4 and described in Section 5.1. It is trained by using training set composed by traces collected as described in Section 4.1. As depicted in Figure 5, decision phase consists in classifying flow (that could be seen as a point in R^n, p^{TS}) to the application of the nearest centroid in R^n. Each centroid has been assigned to a well known application during training phase (c^{new}), according to section 5.1.

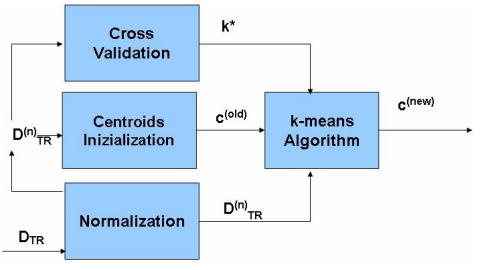

Fig. 4. A k-means based approach: off-line training phase

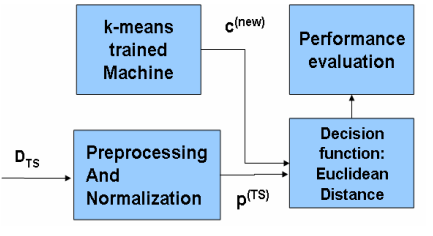

Fig. 5. A k-means based approach: on-line traffic classification

6 Experimental Results

By classifying plain flows with our approach, we obtained results shown in table 1.

Table 1. Plain application flows classification results

		Accuracy (%)				
Pkt <l,d,t>	K*	HTTP	FTP-C	SSH	POP3	Average
3	32	99.0	34.1	99.0	95.7	81.95
4	35	98.9	82.9	98.9	92.3	93.25
5	26	98.6	32.0	99.3	90.6	80.13
6	23	99.2	92.2	98.4	88.3	94.53
7	28	99.5	92.9	99.1	90.2	95.43

We can notice that by analyzing the firsts six packets of each flow we achieve average accuracy up to 94.53%, with 23 clusters as the optimal value found by the cross

validation technique. We can demonstrate that accuracy in recognizing SSH application flows is up to 98.4% relying on the statistical features of lengths, directions and absolute times. We emphasize that increasing the number of analysed packets improves accuracy, even if it delays the classification intended to be real time. Results demonstrate that six packets are necessary to have a trade-off good enough to guarantee high classification accuracy as well as acceptable classification delay.

Once SSH has been classified, we have applied our classification method to identify application flows forwarded in SSH tunnels.

In classifying SSH tunnel content, we have not used arrival times. In fact times related to HTTP over SSH are greater than others due to delay introduced by navigation trough the main server. This could affect our classification results. To make our analysis more realistic for classifying SSH tunnels we have represented each packet using only its size and direction, discharging arrival times.

We tried out processing all possible combination of packets up to ten packets after end of SSH negotiation (i.e. the initial common handshake phase, same in all SSH flows).

Table 2. Encoded SSH applications flows

1°	2°	3°	4°	5°	HTTP over SSH	scp	sftp
0	0	1	1	1	99.80%	98.93%	99.75%
0	0	1	1	0	99.88%	99.30%	99.05%

As shown in Table 2, we tested different patterns representations, increasing the considered number of packets for each flow in order to identify which one contains more information to emphasize difference among applications. As shown in Table 2, the K-means based algorithm yields very interesting results in terms of identification of encoded applications. We can detect different applications with accuracy up to 99.8 for HTTP over SSH protocol, just analyzing third and fourth packets after SSH negotiation. We can notice that analyzing also the fifth packet does not improve significantly accuracy. Moreover, increasing the considered number of packets means introducing delay for real time recognition.

Table 3. Encoded SSH application flows (with scp and sftp upload and download)

3°	4°	5°	6°	7°	8°	9°	HTTP over SSH	scp up	scp down	sftp up	sftp down
1	1	0	0	0	1	0	90.00%	96.78%	95.65%	96.74%	4.61%
1	1	1	0	0	1	0	89.78%	96.83%	95.87%	96.70%	4.61%
1	1	1	0	0	0	1	92.44%	94.65%	88.57%	96.17%	5.61%

We have also tried to classify uploads and downloads of SCP-SSH and SFTP-SSH flows. Results are less encouraging: in fact accuracy decreases for every application. At a glance, SFTP downloads get confused with other applications. By inspecting

data set we concluded that this method mistakes this kind of classification due to the fact that SFTP and SCP are characterized by similar patterns in terms of directions and packet's lengths. Further tests have been performed by analyzing a greater number of packets up to the ninth without achieving significant improvements.

7 Conclusion and Future Works

In this paper we describe a cluster analysis based method to classify in real time encoded traffic flows, overcoming actual limits of deep packet inspection. We are able to identify SSH flows out of other plain TCP based applications with accuracy up to 99.2% after collecting and analysing up to six application packets. Other protocols have been correctly classified with accuracy up to 94.53%. Once SSH flows have been detected, we can classify the nature of each SSH tunnel continuing in gathering session packets. In doing so, we gain accuracy up to 99.88% in classifying HTTP over SSH just analyzing the third and fourth packet after the end of the SSH negotiation phase. The same encouraging results have been obtained by classifying SCP (up to 99.3) and SFTP (up to 99.05) applications. Further works should be performed in order to improve results for classification of download and upload flows for SCP and SFTP. Moreover, it will be necessary to investigate the applicability of the approach on wider application dataset.

Recovering large quantity of well-known traffic is a critical point in traffic analysis due to privacy issues in collecting traces from geographic links. We also tried to classify CAIDA traces [11], but direction of packets is unavailable making our pre-processing assumption useless. We have obtained these good results in our own dataset; yet more traces would help assessing the robustness of our methodology.

Currently on-going work includes extension of the classification tool to more powerful classification algorithms, well beyond k-means; in this respect, k-means shall be regarded as a first use attempt, to verify the soundness of our approach, before proceeding to more complex yet reliable classification algorithms. We are also working on preprocessing and normalization in order to improve the extraction of statistical behavior of application level flows.

References

1. Karagiannis, T., Papagiannaki, D., Faloutsos, M.: BLINC: Multilevel traffic classification in the dark. In: Proc. of ACM SIGCOMM 2005, Philadelphia, PA, USA (August 2005)
2. Crotti, M., Dusi, M., Gringoli, F., Salgarelli, L.: Traffic Classification through Simple Statistical Fingerprinting. ACM SIGCOMM Computer Communication Review 37(1), 5–16 (2007)
3. Wright, C., Monrose, F., Masson, G.: On Inferring Application Protocol Behaviors in Encrypted Network Traffic. Journal of Machine Learning Research (JMLR): Special issue on Machine Learning for Computer Security 7, 2745–2769 (2006)
4. Moore, A.W., Zuev, D.: Internet traffic classification using Bayesian analysis techniques. In: ACM SIGMETRICS 2005, Banff, Alberta, Canada (June 2005)

5. McGregor, A., Hall, M., Lorier, P., Brunskill, J.: Flow clustering using machine learning techniques. In: Barakat, C., Pratt, I. (eds.) PAM 2004. LNCS, vol. 3015, pp. 205–214. Springer, Heidelberg (2004)
6. Zander, S., Nguyen, T., Armitage, G.: Automated traffic classification and application identification using machine learning. In: LCN 2005, Sydney, Australia (November 2005)
7. Bernaille, L., Teixeira, R., Salamatian, K.: Early Application Identification. In: Proceedings of CoNEXT (December 2006)
8. Alshammari, R., Nur Zincir-Heywood, A.: A Flow Based Approach For Ssh Traffic Detection. In: IEEE International Conference on Systems, Man and Cybernetics, 2007. ISIC (2007)
9. http://www.openssh.com/
10. MTU: RFC 879
11. http://www.caida.org