Journal of Information Assurance and Security 4 (2009) 142-150

On the fly Application Flows Identification by exploiting K-Means based classifiers

Gianluca Maiolini[1], Giacomo Molina[2], Andrea Baiocchi[2] and Antonello Rizzi[2]

[1] Elsag Datamat, Automation, security and transportation division,
Via Laurentina 760, Rome, Italy
gianluca.maiolini@elsagdatamat..com

[2] INFOCOM Dept.
University of Roma "Sapienza"
Rome, Italy
Giacomo.Molina@libero.it,
Andrea.Baiocchi@uniroma1.it,
Antonello.Rizzi@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). Moreover encryption of traffic (e.g. VPN) makes ineffective current traffic classification systems based on ports and payload inspection, i.e. Deep Packet Inspection. We have developed a real time traffic classification method based on cluster analysis to identify TCP application flows from statistical parameters, such as length, arrival times and direction of IP packets. By exploiting traffic traces taken at the Networking Lab of our Department and traces from CAIDA, we define data sets made up of thousands of flows of different application protocols. With the classic approach of training and test data sets we show that cluster analysis yields very good results in spite of the little information it is based on, to stick to the real time decision requirement. Moreover, our method works also for identifying applications encoded into SSH tunnels. In this paper we describe our approach and relevant obtained results. We achieved average detection rate up to 95.43% for TCP based application flows and accuracy up to 99.88 % for application flows carried within SSH tunnels, such as SCP, SFTP and HTTP over SSH.

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

1. Introduction

As broadband communications widen the range of popular applications, there is an increasing demand of fast traffic classification means according to the services data is generated by. The specific meaning of service depends on the context and purpose of traffic classifications. In case of traffic filtering for security or policy enforcement purposes, service can be usually identified with application layer protocol. However, many kind of different services exploit HTTP or SSH (e.g. file transfer, multimedia communications, even P2P), so that a simple header based filter (e.g. analyzing the IP address and TCP/UDP port numbers) may be inadequate. So this is also interest in classifying different activities within SSH tunnel or HTTP session. Another example of usage of application traffic classification is differentiated traffic management according to services carried by different traffic flows. A sufficiently robust classifier could be a useful element in implementing differentiated QoS without deploying complex traffic engineering schemes that require cooperation with end hosts.

Traffic classification at application level can be therefore based on the analysis of the entire packet content (header plus payload), usually by means of finite state machine schemes. Although there are widely available software tools for such a classification approach (e.g. L7filter, BRO, Snort), they can hardly catch up with high speed links and are usually inadequate for backbone use (e.g. Gbps links). In addition DPI fails in classifying application when flows are natively encoded. The solution based on port analysis is becoming ineffective because of applications running on non-standard ports (e.g. peer-to-peer). In this scenario we think statistical approach as a direction for effective and reliable traffic classification.

The objective of our work is to develop a real time system to recognize and classify TCP based application flows by analyzing statistical features of first packets of each connection, such as arrival times, directions and lengths. In addition we aim at demonstrate that our method can effectively work classifying SSH encoded traffic flows. 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 not applicable. Actually DPI technology cannot recognize application delivered within SSH flows. By SSH traffic 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 high recognition accuracy up to 94.53% for TCP

Received June 10, 2009

1554-1010 $ 03.50 Dynamic Publishers, Inc

based application flows by analyzing first six packets of each connection. Moreover it provides accuracy up to 99.8% in recognize application tunneled into SSH flows (i.e. HTTP over SSH, SFTP, SCP).

The paper is organized as follows. Section II establishes the position of this work, relative to earlier research. Problem statement is discussed in section III. In section IV we describe dataset creation, in particular data collection and pre-processing. The cluster analysis 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 work

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 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. 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). 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.

We believe that packets-arrival time contains pieces of information relevant to address the classification problem. We provide a way to exalt the information contained in -arrival times, preserving the real-time characteristic of the approach described in [7]. Moreover, we want to realize an effective solution by analyzing client-to-server and server to client directions as well as packet lengths. Our approach relying on statistical features of IP packets can be addressed also for encrypted flows. Other works are focalized on this topic, in particular on SSH. 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 protocol. Our objective is to recognize applications such as HTTP, FTP, POP3 and SSH, by relying on statistical feature of IP packets 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. Ordering is based on timestamp.

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 (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 ε [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 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 according to RFC 793 refers, typical values range between 40 and 56 bytes, depending on options in the TCP and IP headers.

4. Dataset 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. In addition two thousands of POP3 flows were collected by capturing traffic generated by different users of our Lab network who handled mails during several days of work activities. The artificial traffic (HTTP and FTP control) as well as POP3 realistic traces have been sniffed from 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.

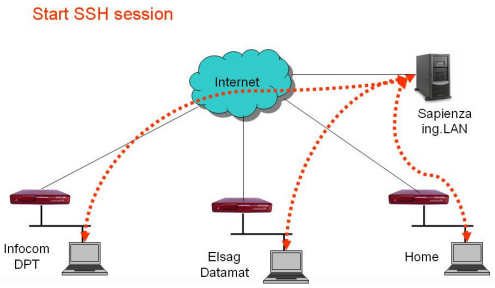

Figure 1 - Platform used to generate SSH traffic: SSH server is inside the University campus network; clients are at University, Elsag Datamat and a private home premise, respectively.

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 removed from each flow F packets related to: first two packets of three-way handshake of TCP: $PK_{0=SYN}$, $PK_{1=SYN-ACK}$, TCP ACK packets, i.e. those packets carrying only a TCP level ACK and no payload data; Retransmitted packets. In order to remove ACK packets and TCP/IP 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:

$\underline{PK_i}^* = <\delta_i=d_j, \lambda_i = l_j-l_{ACK}, \tau_i = t_j-t_{ACK}>, \quad j \in [2,..., N-N_{ACK}-1]$;

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 $\underline{PK_i}^*$ will contain bytes concerning just application contribution. In particular, $\lambda_j \in (0, l_{MTU}-l_{ACK}]$. It is very important underlying the fact that in the case of third packet contains a piggyback ACK, IP/TCP header length (easily detectable from IP/TCP header) will be removed from l_j. In doing so we do not lose application information brought from piggyback ACK third packet.

In order to make our analysis in real time, we need to run our classification method exploiting the very first n packets of each flow. Where n is the number of application packets after the TCP three way handshake. So, we need to analyze different datasets composed by flows of increasing values of n packets. Our dataset will be composed as shown in Figure 2. After tests and analysis of results we set value of n required to strike a convenient trade-off between high classification accuracy and an acceptable classification delay. We collected traces belonging to the application layer protocols: HTTP, FTP-C, POP3 and SSH.

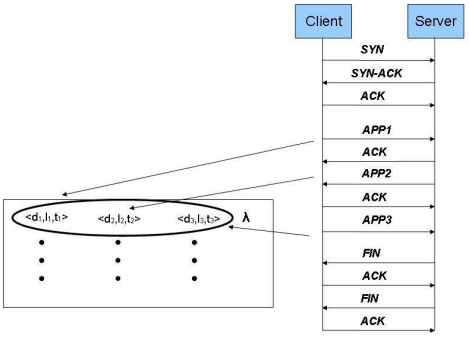

Figure 2 – Data set composed by pre-processing TCP flows, in the case of n=3

Moreover when SSH flows have 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.

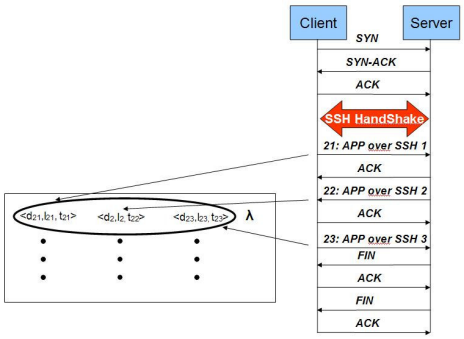

Figure 3 - Pre-processing of SSH flows

As we are indeed interested only on those packets that carry the first few encapsulated segments of the outer connection, we must start collecting packets only after the SSH signaling that triggers the opening of a new forwarding channel is transmitted. To this end we thoroughly analyzed the SSH protocol as implemented in the OpenSSH suite [9] and we verified that the channel open message is always characterized by a 96 bytes long packet sent by the SSH client to the SSH server, followed by a 48 bytes long confirmation packet sent back to the SSH client. What is interesting to note here is that such pair of packets appears only during a channel opening. This helps us removing of SSH handshake packets.

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 $\overline{M}$ of the classification model family M, such that the classification error of $\overline{M}$ 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:

$$\mu_k = \frac{1}{m_k} \sum_{x_i \varepsilon C_k} x_i \qquad (1)$$

One of the simplest, yet powerful, algorithms for K-clustering is the K-Means. This algorithm performs the following three steps:

1. Initialization. It consists in initializing K centroids, by randomly selecting K different patterns in the data set A. At this stage each cluster is empty.
2. Main loop. For each pattern x_i in A:
   a) Compute the closest centroid:
   $$\mu_j = \arg\min_{1 \le h \le k} \{d(x_i, \mu_h)\} \qquad (2)$$
   b) Assign the pattern x_i to the cluster C_j represented by the centroid μ_j.
3. Centroids' update. For each cluster, compute μ_k by eq. (2).
4. Verify stop condition. If a predefined stop condition is true then stop; otherwise go to step 2.

Usually the stop criterion is defined as the logical OR of the two following conditions:

- A predefined maximum number of loops (Epochs) have been performed.
- The average displacements of the centroids between two successive iterations doesn't exceed a predefined threshold:

$$\frac{1}{k} = \sum_{j=1}^{K} \left\| \mu^{(new)}{}_j - \mu^{(old)}{}_j \right\| \le \delta \qquad (3)$$

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.

5.1.1 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:

$$y = \frac{x - \min}{\max - \min} \qquad 4)$$

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.

5.1.2 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:

$$\Pi = \begin{pmatrix} p_{1,1} & p_{1,2} & .. & p_{1,n} \\ p_{2,1} & .. & .. & .. \\ .. & .. & .. & .. \\ p_{k,1} & p_{k,2} & .. & p_{k,n} \end{pmatrix} \qquad 5)$$

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.

$$p_{k,i} = \frac{1}{a} \sum_{h=1}^{a} acc_h^{(i)}(k) \qquad 6)$$

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

$$\pi(k) = \frac{1}{n}\sum_{i=1}^{n} p_{k,i} \qquad 7)$$

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:

$$k = \arg\max_{1 \le k \le l}\{\pi(k)\} \qquad 8)$$

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).

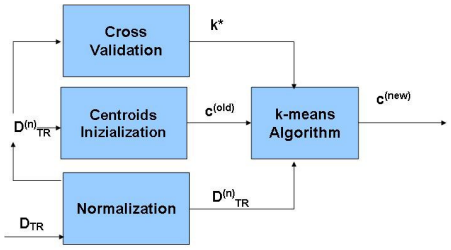

Figure 4 - A k-means based approach: off-line training phase

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.

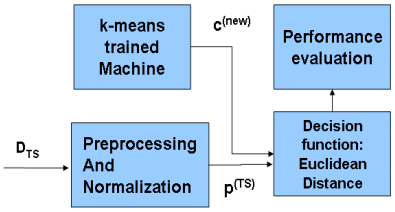

Figure 5 - A k-means based approach: on-line traffic classification

5. Experimental Results

By classifying plain flows with our approach, we obtained results shown in table 1. Test phase of our classification system based on k-means, is repeated for an increasing number of packets (starting from 2 packets). For each application a set of 1000 flows has been inserted in test-set. For each number of packet selected cross validation results give us the minimum number of clusters (K*) necessary to achieve fixed level of accuracy (section 5.1.2).

	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

Table 1 - Plain application flows classification results

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. Complexity of our classification system is proportional to the number of packet analyzed and number of clusters. It is important underline the fact that if number of clusters increase, than accuracy levels will improve, but it will also increase complexity of the classification system. By showing results 23 is the minimum number of clusters. Moreover we emphasize that by 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. So, by choosing to analyze six packets, we set trade-

off good enough between average accuracy and complexity. As shown in results table 1, accuracy in recognizing SSH application flows is up to 98.4% relying on the statistical features of lengths, directions and absolute times.

CAIDA classification results: In this section we present classification results obtained by testing dataset generated from CAIDA public dataset. Results are less encouraging than those obtained in our own dataset (containing reliable traffic traces) due to the fact that for reasonable privacy issue CAIDA traces have packet payloads stripped off, this makes us unable to reliably identify traffic by observing TCP header information. Moreover traces are unidirectional, this imply that our preprocessing tool cannot rely on crucial information contained into packet flows of both direction. Anyway in table 2 are presented the best results obtained relying on arrival times and lengths.

		Accuracy (%)				
CAIDA	K*	HTTP	FTP-C	SSH	POP3	Average
<l,t>	12	70.62	70	83.39	39.85	66.17
<l>	40	72.62	89.69	98.46	75.23	84

Table 2. Caida results

As shown in table 2, accuracy level achieved (up to 84%) proves that packet lengths are useful information to recognize application flows, even analyzing just one direction flows.

Effect of ACKs packets in plain TCP flows detection: In 4.2 we have described the pre-processing made on the collected traffic traces with the aim of highlighting the differences among various applications. In this section we want to verify if this theory is supported by experimental results, in particular we want to verify if removing ACKs packet from traces permit the flows to be characterized in a better way.

In table 3 we compared the confusion matrix obtained by classifying plain flows belonging to a pre-processed dataset (fig 6) with the one obtained by classifying plain flows belonging to a non pre-processed dataset (fig 7). In both cases we have used flows composed of the first five packet of the TCP session (with or without ACKs respectively) remembering that plain flows in our work means that for each packet we report three characteristics: packets lengths, packets direction and packets relative arrival time.

a)	HTTP	FTP-C	SSH	POP3
HTTP	986	13	0	1
FTP-C	27	320	10	643
SSH	0	7	993	0
POP3	3	90	1	906

b)	HTTP	FTP-C	SSH	POP3
HTTP	997	3	0	0
FTP-C	15	982	2	1
SSH	7	8	982	3
POP3	0	99	0	901

Table 3. Confusion Matrixes for the classification of pre-processed flows and not pre-processed flows

This confusion matrixes report on each row the number of flows, of 1000 belonging to a specific application, which are classified to all the four applications. In these matrixes we can read the correct classification results on the main diagonal so the higher is the number of flows the better is the accuracy of the classification calculated as the ratio between the number of flows and 1000.

As we can see, in contrast with our suppositions, the numbers of flows properly classified don't increase if we preprocess the traces. In fact we can observe that the numbers in the principal diagonal are almost similar and even in one case (FTP-Control) the number of flows properly classified increases hugely if we don't preprocess the traces.

These particular results on FTP-Control flows are due to a similar pattern of the TCP session of this application with the one of POP3. In fact if we remove ACKs packet from the TCP session we obtain the same sequence of packet direction for both FTP-Control and POP3 flows as shown in figure 6.

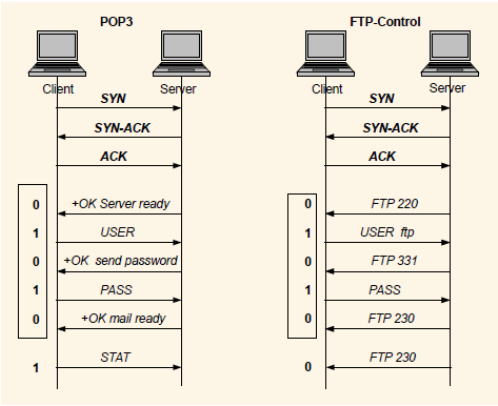

Figure 6 – Sequence of packets direction of a POP3 and FTP-Control pre-processed flows

This fact involves that the flows of the two applications during the training phase of our classification, are going to belong to the same cluster. This cluster is labeled with the label of the application which have the highest number of flows inside it and so during the classification phase we can't distinguish with high accuracy the flows belonging to these applications.

On the contrary if we don't remove ACKs packet from the flows, the packet direction sequence of FTP-Control flows differs from the POP3 one (as shown in figure 7), so the flows during the training phase populate different clusters and the classification has better results.

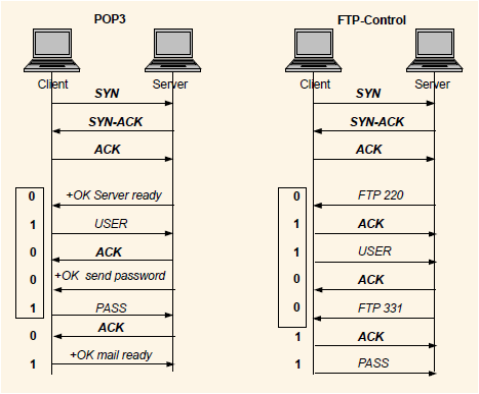

Figure 7 - Sequence of packets direction of a POP3 and FTP-Control not pre-processed flows

These experimental results involve two important remarks connected between them: the first is the presence of ACKs packets always in the same position in the flows packets sequence for the most of the flows of these two classes, so these TCP special packets contribute to the characterization of the application flows and aren't as we supposed confusion elements in this operation; the second is that evidently in the real application technology of internet protocol ACK packets don't depend only on the TCP layer but also on the specific used application, probably because these packets are used to confirm the correct execution of some important operation between the two peers involving in the application handshake. Although, by analyzing six application packets without ACK, so by applying our preprocessing, permit us to distinguish FTP-control and POP3 application, as demonstrated in confusing matrix of table 4.

	HTTP	FTP-C	SSH	POP3
HTTP	979	21	0	0
FTP-C	0	919	12	69
SSH	0	13	984	3
POP3	0	114	1	885

Table 4. Classification result after 6 TCP application packets

SSH results. Once SSH recognition has been accomplished, our classification method has been exploited 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).

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%

Table 5. Encoded SSH applications flows

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.

7 Conclusion and future works

In this paper we present a model that could be useful to address the problem of traffic classification. To this end, we use only (poor) information available at network layer, namely packets size, directions and inter-arrival times. Our classification system based on cluster analysis can classify in real time TCP application flows and encoded SSH traffic flows, overcoming actual limits of deep packet inspection.

We are able to identify HTTP flows with accuracy up to 99.2% after collecting and analysing up to six application packets. Other protocols have been correctly classified with average 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, K. Papagiannaki, M. Faloutsos, ”BLINC: Multilevel traffic classification in the dark”, Proc. of ACM SIGCOMM 2005, Philadelphia, PA, USA, August 2005.

[2] M. Crotti, M. Dusi, F. Gringoli, L. Salgarelli, ”Traffic Classification through Simple Statistical Fingerprinting”, ACM SIGCOMM Computer Communication Review, Vol. 37, No. 1, pp. 5-16, Jan. 2007.

[3] C. Wright, F. Monrose, G. Masson, ”On Inferring Application Protocol Behaviors in Encrypted Network Traffic”, Journal of Machine Learning Research (JMLR): Special issue on Machine Learning for Computer Security, volume 7, pp. 2745-2769, 2006.

[4] A.W. Moore, D. Zuev, ”Internet traffic classification using Bayesian analysis techniques”, ACM SIGMETRICS 2005, Banff, Alberta, Canada, June 2005.

[5] A. McGregor, M. Hall, P. Lorier, J. Brunskill, ”Flow clustering using machine learning techniques”, PAM 2004, Antibes Juan-les-Pins, France, April 2004.

[6] S. Zander, T. Nguyen, G. Armitage, ”Automated traffic classification and application identification using machine learning”, LCN 2005, Sydney, Australia, November 2005.

[7] L. Bernaille, R. Teixeira, and K. Salamatian, ”Early Application Identification”, in proceedings of CoNEXT, December 2006.

[8] R. Alshammari and A. Nur Zincir-Heywood. “A Flow Based Approach For Ssh Traffic Detection”, Systems, Man and Cybernetics, 2007. ISIC. IEEE International Conference on.

[9] http://www.openssh.com/

[10] MTU: RFC 879.

[11] http://www.caida.org.

Author Biographies

Gianluca Maiolini: joined in November 2007 the INFOCOM Dept. of the University of Rome "Sapienza" as a Ph.D. Student in information and communication engineering. He is currently carrying out research for network security, in particular on traffic analysis and packet filtering security devices optimization. He graduated in 2005 in Telecommunications Engineering at the University of Rome "Sapienza". From September 2004 he is employed in ELSAG Datamat S.p.A. in the Automation, Security and Transportation Division where he has been involved in projects related to secure communications, public key infrastructure and secure management systems.

Giacomo Molina: He graduated in Telecommunications Engineering at the University of Rome "Sapienza", in September 2008. He is carrying out research on traffic analysis with INFOCOM department of “Sapienza”. From February 2009 is employed in Smetana S.r.l. and he is now working in the Security Operation Center of Telecomitalia in a project related to systems monitoring and security events correlation.

Andrea Baiocchi: received his "Laurea" degree in Electronics Engineering in 1987 and his "Dottorato di Ricerca" (PhD degree) in Information and Communications Engineering in 1992, both from the University of Roma "La Sapienza", where he is currently Full Professor. The research interests of Andrea Baiocchi are focused on congestion control for TCP/IP networks, on TCP adaptation and packet scheduling over the wireless interfaces, and on network security. He has taken part in many national (CNR, MUR) and international (European Union, ESA) projects, also taking coordination and responbility roles. Andrea Baiocchi has published more than ninety papers on international journals and conference proceedings, he has participated to the Technical Program Committees of more than twenty international conferences; he also served in the editorial board of the telecommunications technical journal published by Telecom Italia for ten years. He serves currently in the editorial board of the International Journal of Internet Technology and Secured Transactions (IJITST).

Dr. Antonello Rizzi: is an Assistant Professor at the Information and Communication Department (INFO-COM) of the University of Rome "La Sapienza" since 2000. His major fields of interest include supervised and unsupervised data driven modeling techniques, neural networks, fuzzy systems and evolutionary algorithms. His research activity concerns the design of automatic modeling systems, with particular emphasis on classification, clustering, function approximation, and prediction problems. In particular, he is currently working on template matching techniques, classification and clustering systems for structured patterns, symbolic inductive modeling systems (Granular Computing) and hierarchical reasoning. He is author or co-author of more than 50 publications. From the current year, he is the Director of the Intelligent Signal Processing and Inductive Modeling Systems Laboratory of the Sustainable Mobility Research Center of the University of Rome. He received his Dr. Eng. degree in Electrical Engineering in 1995 and his Ph.D in Information and Communication Engineering in 2000 from the University of Rome "La Sapienza".