INTERNATIONAL JOURNAL OF NETWORK MANAGEMENT
Int. J. Network Mgmt 2015; **00**:1–14
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/nem

Internet Traffic Classification Based on Flows Statistical Properties with Machine Learning

Alina Vlăduţu[1], Dragoş Comăneci[1] and Ciprian Dobre[1]*

University POLITEHNICA of Bucharest, Faculty of Automatic Control and Computers, 313 Splaiul Independenţei, Sector 6, 060042, Bucharest, Romania

SUMMARY

Machine learning has recently entered the area of network traffic classification as an alternative to the deep packet inspection technique. It provides both unsupervised and supervised learning algorithms that are capable to put aside similar types of traffic or recognise Internet protocols based on some training, pre-labeled samples.
The current work proposes a new approach in the area of network traffic classification using machine learning. First we extract the uni- and bidirectional flows from a traffic capture. A flow is a collection of packets that share sender and receiver IP address and port. Secondly, we select relevant statistical properties of these flows and use a unsupervised learning mechanism to group flows into clusters based on the similarities. Eventually, we use this classification as training input for a supervised learning engine that will have to properly determine the class of new, unseen traffic flows. Copyright © 2015 John Wiley & Sons, Ltd.

Received . . .

KEY WORDS: traffic classification; machine learning; network flows; statistical properties

1. INTRODUCTION

Today's networks are increasingly busy having to support more and more traffic. There are currently three approaches for this [1]: scaling the network horizontally, vertically or both ways. Scaling vertically means that the network is built with more powerful machines able to route more traffic, while scaling horizontally implies bringing more machines of the same power as the current ones. As the number of connections in the networks increases, the more routing paths increase the overall network performance. Both of these approaches require more hardware and, of course, come with a significant cost of acquisitions and maintenance.

A new solution, which recently became popular with various research groups [2, 3, 4], is using machine learning techniques to classify network traffic in order to detect any traffic patterns or anomalies. Detecting the traffic changes is an important operation to adjust the resources accordingly (i.e. changing a routing policy in case a link is congested or ban traffic that come from a malicious IP address) with no additional costs.

The current work aims to explore this area and proposes a new way to classify traffic based on the packets statistical properties using machine learning techniques. We define what a flow is in the following Sections, but the idea is to extract statistical properties of the packets that are flowing through the network and group them together in clusters based on similarities. This is achieved with the aid of K-Means, an unsupervised learning algorithm. Then we proceed to use this classification

*Correspondence to: University POLITEHNICA of Bucharest, Faculty of Automatic Control and Computers, 313 Splaiul Independenţei, Sector 6, 060042, Bucharest, Romania. Email: ciprian.dobre@cs.pub.ro along with all the statistical properties to train a supervised learning engine using C4.5 that will eventually be able to properly classify new traffic.

The classical approach [5, 6] in network traffic classification is deep packet inspection (DPI). It entails that every packet should be checked against the available traffic signatures. It offers fine grained results because the entire package is analysed, not only the header. It is true that high accuracy could be achieved for identifying network protocols and applications but it turns out that the process is a greedy resource consumer. Even though deterministic finite automata (DFA) that are used to parse the regular expressions are fast, they may involve prohibitively large amounts of memory for pattern matching in network applications.

Moreover, deep packet inspection is useless in case of encrypted traffic (i.e., protocols like SSH or HTTPS). The packets cannot be decrypted and everything appears as gibberish. In case of a new virus for which the network equipment does not already a signature, DPI may misbehave and treat the traffic as unknown. The update of traffic signature may be tedious and error prone for a network administrator. The routers and the firewalls are usually the network pieces that are doing the DPI task. In case of a network congestion they could be overwhelmed by the amount of traffic that has to be processed and could bring significant latency. As described above DPI is really expensive in terms of memory used. On a long term basis, if network traffic grows, the costs of doing DPI may increase significantly. These are the main disadvantages of DPI that make it impractical in some situations.

One cannot give up classifying the traffic inside its network. A few reasons why the network traffic should be careful analyzed:

- detecting malicious traffic that may be harmful to network and users; clients/resource access may have to be restricted.
- detecting a growth of traffic; more resources may be brought up or plans for traffic scaling and optimization should be elaborated.
- detecting new traffic patterns; the routing decisions/QoS (quality of service) rules may be altered.
- detecting new traffic trends; better understand the needs of the users.

Given the disadvantages of deep packet inspection shown above and also the need of classifying the network traffic, a new approach in traffic classification will be introduced in Section 2. It involves no additional hardware and minor costs as the current equipment is used with a new software paradigm. It uses both unsupervised and supervised machine learning techniques. In the current paper we propose a new direction in network traffic classification: we extract traffic flows and cluster them using an unsupervised machine learning algorithm. This classification is then used to train a supervised learning engine that is, in the end, able to properly classify new, unseen traffic. The network flows were clustered with and accuracy of 85% while new flows were 90% correctly classified.

The rest of the paper is organised organized as follows: Section 2 presents existing algorithms and approaches in network classification, Section 3 shows the design of our solution along with some experimental results provided in Section 4. Eventually, Section 5 draws a few conclusions.

2. RELATED WORK

First we would like to introduce the definition of a network flow in a TCP/IP based network as mainly used in this work. It is a five-dimension tuple having the following fields:

<source IP, destination IP, source port, destination port, transport protocol >

The transport protocol could be either TCP or UDP. A flow is basically a client-server communication or a dialogue between two peers. We consider this concept important as it is close to the traffic patterns in real networks that we want to classify more accurate. For example, a HTTP client-server communication is a bidirectional flow that is initiated by the client. Authors in [7] analyse the traffic patterns inside a datacenter and define the concept of flow. They find that more than 80% of the datacenter flows last less than 10 seconds.

One usage of machine learning in the context of network traffic is for traffic classification. This way to difference between multiple flows of traffic (based on the network protocol/type of traffic sent) was developed as an alternative to the previous deep packet inspection (DPI) and port matching. DPI analyzes the whole content of every packet and, with the aid of some regular expressions, it tries to find out if it matches some already known type of traffic. Port matching is a trivial way of identifying traffic protocols/applications based on the port they are using. This assumption is based on the well-known ports (i.e., HTTP uses port 80) and application specific ports (BitTorrent uses a port between 6881-6889) but any application can use a non standard port. In the current work we are calculating some statistical properties of the packets that are part of the same flow and try to cluster them based on the similarities they share. Next, this output should be used as training for a supervised learning engine that should be able, to classify unseen traffic in a right way.

Considering previous work ([8]), the observable statistical properties of a flow that could be of interest for study in the context of machine learning are:

- duration of the flow
- total number of packets involved
- packets length taken individually or in total
- flow length in bytes
- inter-packet arrival time

Classifying network traffic using machine learning seems promising. Miller et al. [9] reports an accuracy of 89% for classifying HTTPS web pages coming from the same website in the context of examining the vulnerability of HTTPS to traffic analysis attacks. This is a good result even though they focused more on caching and user-specific cookies and did not take into account details like browser differences, operating system differences, mobile/tablet devices or network location.

2.1. *Machine learning tools for traffic classification*

The machine learning techniques can be split into three categories based on the amount of labeled traffic used as training input: unsupervised learning, supervised learning and semi-supervised learning. Next, we discussed broadly the first two, as they are more importance for the current work: we know nothing forehand about the possible types/groups of traffic so we need to use an unsupervised learning algorithm to cluster the flows and put them together based on affinity. Then, based on this classification, we present our solution to train a supervised learning engine so that it will be able to properly classify new traffic.

2.1.1. Unsupervised learning *Unsupervised learning* has no labeled input data and tries to find any hidden properties inside it. This is achieved using some techniques that we describe next.

Clustering is the process of grouping the inputs into a given number of k groups based on resemblance. One group of input data that share a common pattern is called a cluster. There are a few reasons [10] why it would be useful to have this:

- Underlying structure: to get an insight on how the data looks like, to detect features or anomalies
- Natural classification: to identify similarities between different organisms
- Diversity of clusters: to identity groups based on different criteria
- Compression: to organize data based on the cluster prototypes

For the current work, clustering the network flows is a key step as we try to put aside different protocols/types of traffic based only on the statistical properties of the flows or of the packets that are part of that flow. We are going to vary the number of clusters and learn which brings the most accurate classification of the input.

Dating back in the last 1950s, *K-Means* is a hard-assignment problem formulation that aims to assign every input data in only one cluster. This is opposed to the soft-assignment that proposed in late 1980s, and where every data-point could be part of multiple clusters each with its membership value. To formalise[†] it, given a n input data points $(x_1, x_2, \ldots, x_n)$, each being a d-dimensional array of values, the K-Means problem aims to partition the n points into $k(n)$ sets $S = S_1, S_2, \ldots, S_k$ so as to minimize the within-cluster sum of squares.

There are many implementations ([11, 12, 13]) of this algorithm. One of the difference between them consists in the way they handle replacing the classical way of calculating distances to the clusters with some other heuristics, like the Lloyd's algorithm ([14]) or k-Means++ ([15]). For our implementation, we use the Weka [16] built-in K-Means algorithm. Our decision is based on the fact that this is one of the simplest approach, and the precision of K-Means offered is comparable with the ones obtained when using expectation maximization and random forest proximities (described next).

Next, the *expectation maximization* (EM) [17] algorithm was primary used in the biology field of study, for probabilistic models such as hidden Markov models or Bayesian networks. However, it is also used for data-to-cluster assignment, enabling missing data estimation in case of an incomplete data set. It repeats the following two steps:

- Estimation (the E-step): given the current model, it tries to complete the data that is missing by guessing a probability distribution
- Maximisation (the M-step): using the previous completions, it re-estimates the model parameters

More recently, researchers demonstrated the power of the *Random Forest Proximities* [18] algorithms, as an alternative to the well-known methods for network traffic classification (for a comparison between Random Foresc Proximities and K-means or EM, we refer the reader to [19]).

Random Forest [20] is a supervised learning algorithm that was proven to show great results in traffic classification [21]. It also shows up useful features that leads to an increase in the accuracy of results: unlike other methods, Random Forest allows as input to have data points clusterization, and can estimate the generalization error during the learning process and/or calculate the proximity distance between all input pairs. The *Random Forest Proximities* consists of two major operations that will be discussed further: building a forest out of the input unlabeled data, and the clustering of data based on the proximity matrix built using the K-Medoids method.

A random forest is a collection of multiple classification trees. In order to classify a data point, each tree predicts a class (votes for a class). The class that has the most votes inside the forest is the predicted class. Two important input numbers are important for building the forest: the number of trees and the number of variables used for splitting the nodes. The latter should be significantly smaller than the number of input values.

Each tree is built from root to leaves and should be as large as possible without any pruning. The root node is first created by taking N random sample data points with replacement from the training set. N is the size of the training data set. After this, the nodes are iteratively split using the Gini index criterion based on m variables that are randomly selected from the input variables. After building the forest, the proximities for every pair of points can be computed. If two points fall in the same leaf node, their proximity is increased by one. Eventually all proximities are normalised and divided by the number of created trees. A symmetric proximity matrix is created where the proximity between a point and itself is set to 1 and all the other proximities have to be between 0 and 1.

In case of classifying network traffic, the input data is not labeled so the forest cannot be built directly. To solve this, a synthetic classification problem is first defined in order to distinguish the original data from the synthetic data. The final forest and the proximity measures will hardly depend on the synthetic samples. As the authors of [20] show, a second class is created by sampling at random in each dimension according to the empirical univariate distributions of the original data. Thus, the synthetic class has a distribution of independent random variables and each of them follows the same distribution as the corresponding variable in the original observations. The process

[†] `http://home.deib.polimi.it/matteucc/Clustering/tutorial_html/kmeans.html`

actually breaks the structure of the original data. If the misclassification rate is low it means that the features dependencies are quite low.

When the forest is built we build the proximity matrix by running the original unlabeled data. The process has to be repeated a number of times (i.e., T, the number of forests) because, as the name suggests, the random forests are random and so is the proximity matrix. Consequently, the final result will be the average of all T results.

The proximity matrix is further used as an input for dissimilarity-based clustering engines, like the Partitioning Around Medoids. K-Medoids clustering algorithm, though similar to K-Means, has two main differences:

- it tries to minimize the sum of pairwise dissimilarities between data points and the cluster centers instead of using the squared Euclidian distances;
- it allows measuring arbitrary distances between data points instead of making use of the Euclidian distance.

2.1.2. Supervised learning In contrast to the unsupervised learning that receives no training input, *supervised learning* is a machine learning technique that tries to infer a function based on a pre-labeled training set. This function is further used to determine the class or the value associated to any new, unknown data. Based on the nature of the value associated to the input "key", there are two approaches: classification that uses discrete output values and associates to every input a label and regression that has a continuous range of output values and supplies a result. Classification is what we are doing in the current work so it will be described briefly.

Statistical *classification* is an example of supervised learning that provides a class for a new, unseen input based the training, pre-labeled data set. One way to do this is by building a decision tree as the following algorithms are doing.

The *Iterative Dichotomiser 3* (ID3) algorithm is the precursor of C4.5 in the field of decision tree machine learning. ID3 is starting from a set of sample data that has to be in attribute-value style. The values of the attributes has to be discreet and already defined, as the algorithm is not capable of learning any new thing. It starts to iterate through the unused attributes and calculates the entropy (or information gain) for each of them. The entropy is a number between 0 and 1. 0 here means "perfectly classified", and 1 means "totally random". The attribute with the smallest entropy is next selected. The root node S consisting of the original set will split by this attribute (i.e., $weather = rainy$, $weather = cloudy$, $weather = sunny$) to produce subsets of data. The algorithm will continue recursively on each subset, considering attributes that have not been previously used. Eventually, a decision tree is used to classify new, unseen traffic. ID3 does not guarantee an optimal solution, and is also prone to overfitting the training set by building a very big decision tree. Small trees are preferred in many cases.

The *C4.5 algorithm* is similar to ID3, but brings several improvements. It accepts both discrete and continuous attributes. To do so, it splits the continuous interval into values that are above and below a certain threshold. C4.5 accepts missing attribute values. They are marked with a "?" (question mark) and are not taken into account when computing the entropy or information gain. It can also handle attributes with different costs, and finally, after the decision tree is built, the algorithm is performing a pruning operation by removing unuseful branches. Various author ([22, 23]) proved that C4.5 is the fastest decision tree based supervised learning technique for traffic classification, and for this reason we decided to also use it. During the experiments for this work, we will use the J48 open source Java implementation of C4.5 that is integrated in Weka[‡].

2.2. *A critical analysis of related work in network traffic classification - key findings, challenges and limitations of existing work, proposed research direction*

Classifying the network traffic using machine learning techniques gained attention into the research community as a promising way to cope with problems that port-matching and deep packet inspection

[‡]http://weka.sourceforge.net/doc.dev/weka/classifiers/trees/J48.html

have, like being computational expensive or hopeless when dealing with tunneled, obfuscated or encrypted traffic. Authors in [24] present a critical analysis and comparison of methods for traffic classification, and indicate possible future development directions.

The analysis in [24] also draw attention to the multitude of different terminologies and granularity of definitions that make the evaluation very hard even when using the same metrics and data set. Different granularities objects could possibly have:

- TCP connections identified based on the TCP flags or TCP state machines;
- flows (unidirectional flows) and/or biflows (bidirectional flows) with the same meaning we introduced in this work;
- applications (i.e. POP, IMAP, SMTP) or application classes (i.e. mail);
- coarse grained content type (i.e. text, binary, encrypted traffic) vs fined grained content type (i.e. text, picture, audio).

The terminology is one of the issues the future researchers has to deal with. Moreover, an important problem is the lack of shareable test data input along with re-labeled flow objects to be used as a reference. Privacy and ownership constrains when disclosing the data, make the mission of researchers hard in their attempt to improve the way traffic is classified. They actually encourage, inside dedicated communities, the development of tools that annotate the training data with a class. They also find important the capacity of building scalable, parallel traffic classifiers that will be able to deal with the huge amount of data that is coming in the context of networks being more and more powerful.

Focusing now on a few more specific unsupervised and supervised traffic classifiers, Simple K-Means is used in [25] to classify TCP-based application bidirectional flows using only the dimension of the first P packets. The assumption of using only a few first packets is that every application, during the negotiation phase at the beginning is using a certain, pre-defined sequence of messages that makes it unique between other applications. There is the offline training phase that involves the clustering of the P dimension bidirectional flows. Simple K-Means is using the Euclidian distance to determine which flow belongs to which of the k clusters (k is provided as parameter). The output is the description of the cluster (along with its center) and the applications it contains. This piece of information is used in the second phase, the classification phase, that is done online. The results are impressive: more that 80% of the flows were correctly classified and stimulate the early detection of applications. One problem that is not covered in [25] is the presence, in the offline phase, of some flows that are not caught from the beginning. This inspired us to also use Simple K-Means from Weka with reasonable expectations and also explore some new statistical properties of the packets/flows.

Authors of [8] use supervised learning techniques for traffic classification. It uses as training flows that were previously labeled by hand with a certain class which is a more coarse grained one: database, games, mail, multimedia, P2P, etc. Statistical properties like flow duration, TCP port, packet inter-arrival time (mean, variance), payload size (mean, variance), effective bandwidth-based entropy ([26]), Fourier Transform of the packet inter-arrival time, are all used to classify the flows using a more probabilistic approach with the Native Bayesian classifier. Using the most basic form of a Naive Bayes classifier, accuracy rates of around 65% were obtained ([26]), while with the addition of some improvements (Bayes based upon kernel-estimates combined with the fast correlation-based filter (FCBF) technique for discriminator reduction) the accuracy raised to 95%.

Based on the previous works described, *we propose a new research direction in traffic classification which combines unsupervised and supervised learning techniques*. We use statistical properties of the packets that were already used by others with good results, but we will also use Simple K-Means and C4.5 (among most efficient for decision tree based supervised learning [22, 23]) and others.

3. DESIGN AND IMPLEMENTATION

In the next Sections we discuss the requirements of our classification approach, starting from the drawbacks of deep packet inspection and the existing literature in traffic classification with machine learning. An architecture along with the implementation details are further discussed next.

3.1. Design challenges and requirements

The classical way of classifying network traffic is by doing deep packet inspection through a firewall or an intrusion prevention system (IPS)/intrusion detection system (IDS) device. The network device should periodically be updated with new traffic patterns/signatures. This operation is especially important when trying to filter malicious traffic containing viruses, malware or other harmful traffic. Just because illegitimate traffic can take new masks overnight, the traffic signature update has to happen as quickly as possible. This job can be tedious for a network administrator and can also lead to errors. Moreover, the maintenance of the network equipment involved in deep packet inspection can be hard and expensive. And, to make things worse, some kind of traffic cannot be inspected. Packets containing SSH or IPsec traffic are encrypted and obfuscate the inspection operation.

Recent research studies showed that machine learning can help leveraging the process of classifying the network traffic. Statistical properties of the communication between two points can be used to classify traffic and detect anomalies or network changes. An analysis of statistical properties of the network can be done on the flight, without a priori knowing that a certain pattern leads to a certain conclusion. The approach of analysing startical properties is used by various router, needing to detect congestions or network attacks. When an event is detected, routers re-route packets on other paths (or even drop the packets). More advanced techniques, such as deep packets inspection, are advocated lately as leading to even more accurate detection results, but they can also be time consuming, and require accurate models of the network flows. Finally, a machine learning approach can determine automatically traffic pattern changes, and can lead more quickly to network controller decisions.

At the moment, classifying network traffic using exclusively supervised/unsupervised machine learning together with statistical properties of the traffic flowing in the network, was shown to lead to accuracy levels between 50% to 70%, but previous work showed even cases where the accuracy went up to 95% [8]. In this work, we use some of the statistical properties previously used by other authors, but we combine them with new ones that we introduce in the next Section. The scope is to propose a solution designed to improve the classification accuracy by employing machine learning techniques.

3.2. Proposed architecture

During current work, the idea is to first group network traffic flows into clusters considering the statistical properties of flows, in order to avoid deep packet inspection. This clusterization is further used by a supervised learning algorithm to classify new traffic flows.

We start from a network traffic packet capture represented by a .pcap file and extract unidirectional and bidirectional flows. A pcap is a file format for dumping packet traces from tools such as Wireshark, one popular network protocol analyzer. A flow is a 5-dimension tuple: (*source IP, destination IP, source port, destination port, transport protocol*). Every flow is composed of packets that are going in one (from A to B) or two (from A to B, and back from B to A) directions.

For the unidirectional flows we calculate the following statistical properties:

- the number of packets;
- the duration of the flow (the time between the last and the first packet sent);
- the total length of all/the first ten packets;
- the minimum/maximum/average/standard deviation packet length;
- the minimum/maximum/average/standard deviation inter-packet arrival time.

For the bidirectional flows we concentrate on the following properties:

- the duration of the flow (with the same meaning as in the unidirectional flows);

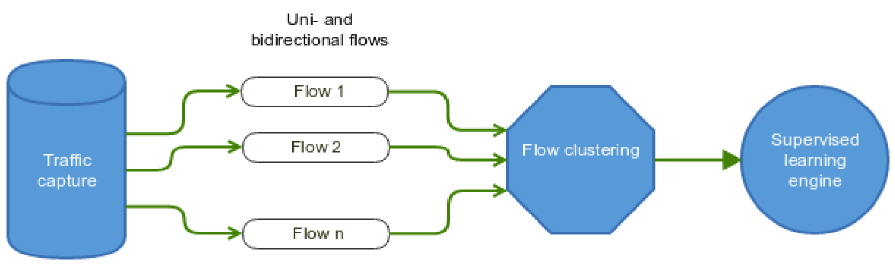

Figure 1. The project design.

- the first ten senders (which peer sent each of the first ten packets);
- the first ten inter-packet arrival times;
- the first ten packets length;
- the first ten differences between inter-packet arrival time;
- the number of packets sent;
- the minimum/maximum/average/standard deviation packet length;
- the minimum/maximum/average/standard deviation inter-packet arrival time.

Authors of [8] use the following statistical properties and show good results: Flow duration, TCP Port, Packet inter-arrival time (mean, variance), Payload size (mean, variance), Effective Bandwidth-based upon entropy ([26]), Fourier Transform of the packet inter-arrival time. We decided to use some of these and try some new features. As most of the flows are TCP flows, the very first packets that are sent between the two peers have some interesting patterns, so we considered taking into account properties that relate to the first ten packets of the flow. Authors of [25] show that every application has a well defined sequence of messages that distinguishes it among the others. Selecting the most relevant statistical properties of the flows is an important step in our effort, because it influences the way packets are grouped into clusters. Starting from this, the machine learning algorithms will come into action: based on the properties defined above, an unsupervised learning algorithm groups the flows into clusters and after that the supervised learning engine takes this classification as input to further identify whether other traffic has one label or another. The whole process is described in Figure 1. The main three steps of our algorithm are shown in Listing 1.

Listing 1: The proposed traffic classification algorithm

```
(1) Extract uni- and bidirectional flows from a traffic capture
based on statistical properties.
(2) Group the above flows into clusters, varying the number
of clusters.
(3) Train a supervised learning engine according to what was found
at the previous step to classify new flows.
```

3.3. Technologies and implementation details

The current implementation involves, beside other technologies, only code written in Java. At first we generate a traffic capture, expressed as a .pcap file using Ixia BreakingPoint application traffic emulator[§]. The tool is capable of simulating various types of traffic that can be seen in the network

[§]http://www.ixiacom.com/products/ixia-breakingpoint

(HTTP, Youtube, IMAP, SSH, FTP, etc.). Each application flow is simulated based on the protocol specification, and can have multiple random payloads. We will deep dive into more details about this in Section 4, when we present experimental results.

In order to extract packets from the traffic capture and build the uni- and bidirectional flows, we added in Eclipse, our working integrated development environment (IDE), a dependency to jpcap, a network packet capture library as a .jar file¶. The library helps to parse every packet in the capture and extract different properties: source IP, destination IP, source port, destination port, transport protocol (TCP/UDP) and its length. After getting this information, we determine whether the packet is part of an existing unidirectional or bidirectional flow. If so, we recalculate the flow properties or save the length of the packet for future use (i.e. standard deviation packet length has to be calculated at the end when we have all lengths of the packets available). If the packet is not part of an existing flow we create two new flows, one unidirectional and one bidirectional, to reflect the flows this packet is part of.

After parsing of the .pcap file is over, and all final data processing is done, we write the output in two text files, one for each type of flows. Each flow is written on a line. On every line the 5-dimension tuple flow identifier is written first, and then come all the other properties recently calculated. All values are separated by comma.

We then proceed to run the K-Means in order to classify network flows.We are using Weka for this, a Java based machine learning framework. We used its facility of Java code integration via library, and added a .jar dependency to our project. The above text output containing the properties of the flows will be used as input for K-Means but it has first to be converted to the arff internal format of the tool. The arff file type is the standard input type for both Weka GUI and Weka Java integration. An arff file is also a text file that has to describe at the beginning the name and type of the columns that are going to appear on each row of the data input.

The run of K-Means receives the number of clusters it has to produce as parameter. We will vary this while measuring the testing results. The algorithm will produce two pieces of output. First, a text file that contains, for each cluster, the number of flows classified as being part of it, along with a percent from all flows that were labeled so. Second, a new arff file, similar to the previous one with the addition of the cluster number each flow was distributed into, is written.

The last arff file is further used as training input for the C4.5 algorithm in Weka GUI in order to classify new traffic. The C4.5 implementation in Weka is J48, and is open source. The results obtained are next presented in Section 4.

4. EVALUATION RESULTS

In the following we present results obtained by our application during several tests. We first describe how the testing sample was obtained, followed by an analyse of the results for multiple number of clusters for the K-Means classification. Then, we show how the supervised learning engine behaves when the input is received from the previously described step.

4.1. Experimental setup and working hypothesis

For all our experiments we used a 2GB traffic capture that was generated with Ixia BreakingPoint‖ traffic emulator application. The tool is able to simulate real-life application traffic that could appear in every enterprise network. Each type of application flow is emulated according to the protocol specification. Every application could have multiple randomized payload sections. The content of the traffic capture contained in a pcap file has 14 types of enterprise application traffic, and is depicted in Table I.

All the experiments described were conducted on a machine with the following specification: Intel Core i5 CPU, 4GB RAM, 128 GB SSD, Ubuntu 14.04.3 LTS operating system, Java

¶http://jpcap.sourceforge.net/
‖http://www.ixiacom.com/products/ixia-breakingpoint

Traffic type	Percentage from the whole traffic
HTTP Video Enterprise	29.50%
HTTP Enterprise	18.63%
SSH Enterprise	17.96%
Oracle Enterprise	17.15%
Raw UDP Enterprise	5.96%
Raw Enterprise	3.71%
BitTorrent Enterprise	2.09%
Flash Enterprise	1.93%
HTTPS Simulated Enterprise	0.90%
SMB Enterprise	0.75%
SMTP Enterprise	0.56%
PPLive Enterprise	0.51%
FTP Enterprise	0.32%
YouTube Enterprise	0.03%

Table I. Types of traffic in percentage in our experimental capture.

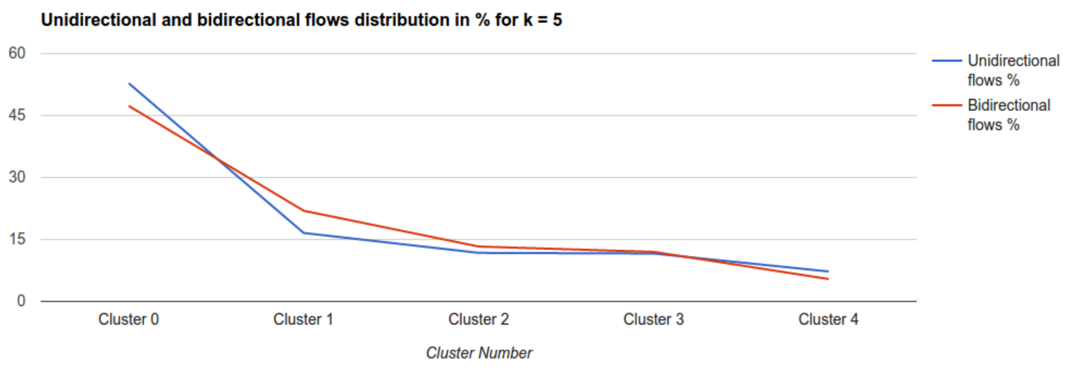

Figure 2. Unidirectional and bidirectional flows distribution in % for k = 5.

version 1.7.0_80. The unidirectional and bidirectional flows, along with their statistical properties described earlier, are extracted in 172.573 seconds. There are 100748 unidirectional flows and 53956 bidirectional flows in the current file. Next, we proceed to clusterize both types of flows with K-Means and vary the k number of clusters we want to have. Here, k takes the following values: 5, 10, 15, 20. Based on the input capture and the number of different types of traffic it contains, we decided to pick up these values to see how the precision of the clusterization evolves. Empirically, we expect the most accurate clusterization to happen for $k = 15$. The intuition is that the number of clusters should be as close as possible to the number of traffic types in the capture, as the clusterization step is the most important step in our effort of classifying the traffic based only on the statistical properties of the network flows.

4.2. *Results and key findings*

For $k = 5, 10, 15, 20$ we obtained the results for unidirectional and bidirectional flows, shown in Figures 2, 3, 4, and 5.

While the number of clusters grows, the flow distribution is more and more crumbled. This behaviour is somehow natural as the more groups we have, the difference between flows is more subtle. For bidirectional flows, values tend to be less crumbled, partly because there is more information taken into account during classification, and so the process tends to be more accurate.

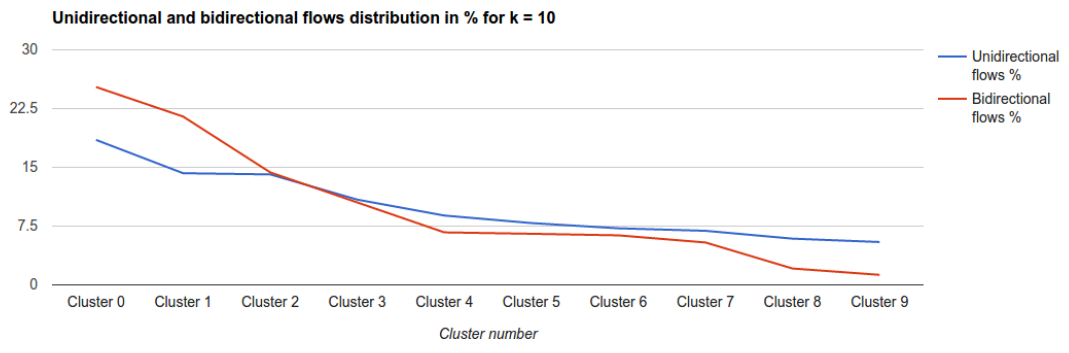

Figure 3. Unidirectional and bidirectional flows distribution in % for k = 10.

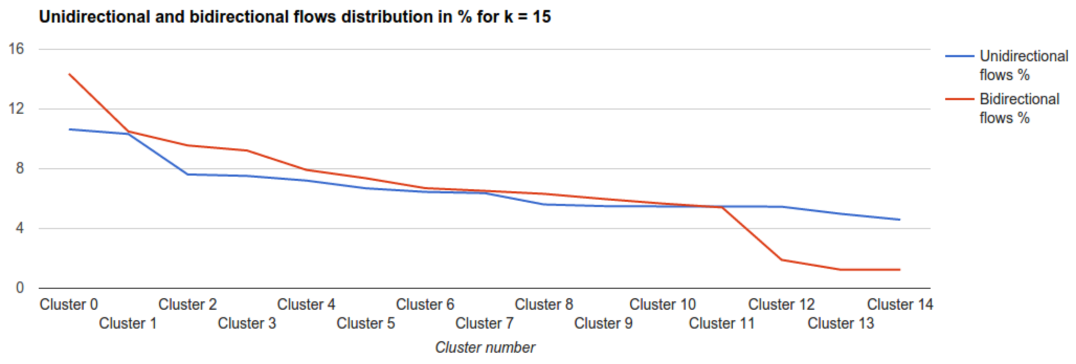

Figure 4. Unidirectional and bidirectional flows distribution in % for k = 15.

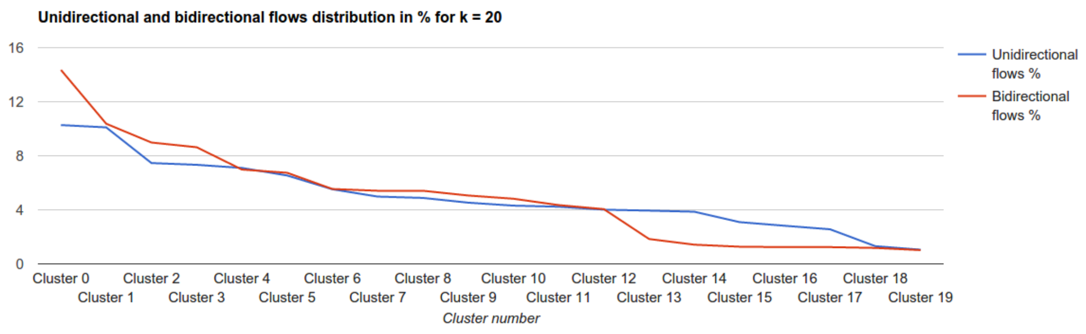

Figure 5. Unidirectional and bidirectional flows distribution in % for k = 20.

Next, we compared the results obtained for the four values used for k. We used the *ttest2*[**] function from the statistics package of Octave (the open source alternative for Mathlab), as follows:

[h, p, ci, stats] = ttest2(x, y, 'Tail', 'both', 'Vartype', 'unequal')

The function tries to test the null hypothesis that the data in vectors x and y comes from independent random samples from normal distributions with equal means and unequal and unknown variances, using the two-sample t-test[††]. The result h is 1 if the test rejects the null hypothesis at the 5% significance level, and 0 otherwise. x is an array that contains the 14 values for the protocol distribution in our test traffic capture from Table I. y is an array that contains the flow to cluster distributions in percentage for unidirectional and bidirectional flows for all the four test cases

[**] http://uk.mathworks.com/help/stats/ttest2.html
[††] http://uk.mathworks.com/help/stats/ttest2.html#btrkaaw

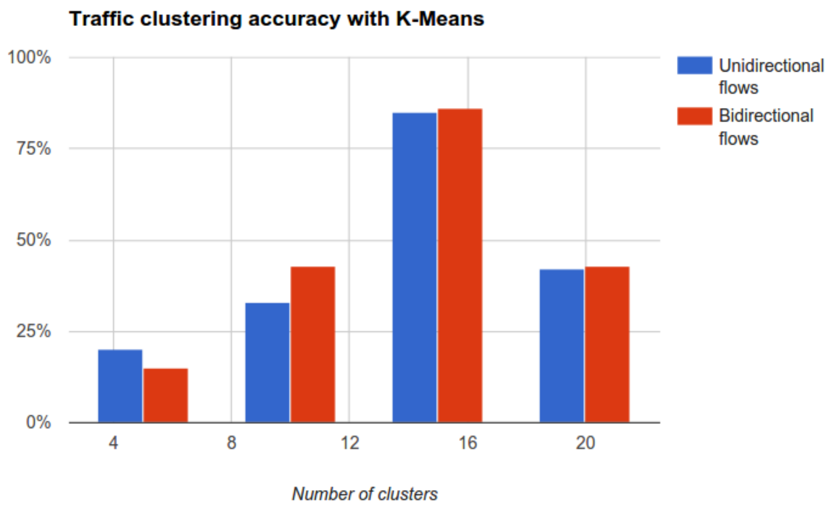

Figure 6. Clustering accuracy with p based on unidirectional and bidirectional flows

($k = 5, 10, 15, 20$). '*Tail*' and '*Vartype*' are name-value pair arguments. 'Tail' means 'both' tests, as in the alternative hypothesis that the population means are not equal, whether 'Vartype' is 'unequal', meaning that the test should be conducted using the assumption that x and y are from normal distributions, with unknown and unequal variances. ci (confidence interval) returns the difference in population means of x and y, while $stats$ is a statistical structure that contains: $tstat$ (value of the test statistic), df (degrees of freedom of the test), and sd (pooled estimate of the population standard deviation, for the equal variance case, or a vector containing the unpooled estimates of the population standard deviations, for the unequal variance case). But the most important piece of output for us is p, which returns a scalar value in the range $[0, 1]$. p is the probability of observing a test statistic as extreme as, or more extreme than, the observed value under the null hypothesis. The higher the p value, more likely that the two samples are similar. The p results encountered during our tests are presented in Table II.

Number of Clusters	Unidirectional flows	Bidirectional flows
5	0.20	0.15
10	0.33	0.43
15	0.85	0.86
20	0.42	0.43

Table II. Clustering accuracy for unidirectional and bidirectional flows for multiple number of clusters

Translating the $[0, 1]$ values of p into percentage, we show in Figure 6 the evolution of the clustering accuracy based on the two types of flows. p is strongly connected with k and with the actual number of clusters inside the capture. When k and the real number of clusters are close, p tends to have higher values, while having values for k that are either too small or too big compared to the actual number of clusters makes p drop to low values (close to 0.2). Our hypothesis is confirmed: the closer is k to the actual number of protocols in the capture, the more accurate the results are. As we can see for $k = 15$, we have values of 0.85 and 0.86 for p. This is why this value of k could be considered a local optimum. One important conclusion we have at the moment is that the election of k is a decisive step. This could be done using some approximations, considering the characteristics of the network and some inside knowledge. We also notice that bidirectional flows tend to have slightly better results. This is because they also use more information regarding the statistical properties of the packets involved in the flow.

We used the Weka K-Means output for $k = 15$, which turned out to be performing the best as an input for the C4.5 algorithm. Weka GUI was used this time that offers the J48 open source implementation for C4.5. The results in case of using 10%, 30%, and 50% of the input for testing and the other part for training, show that more than 90% of the flows were classified correctly. It means that the classification of the new traffic strongly relies on the results from the clustering operation of K-Means which turns out to be the critical operation in the whole process. So k, the number of clusters, should be chosen such that it is as close as possible to the number of application types that go through the network at the time of the analyses.

The results obtained will be further used with the goal of creating an intelligent network control system that can learn from the traffic in the network, and predict and adapt to similar traffic patterns that may appear in the future. This is linked to Software Defined Networks (SDN), as the main problem with SDN controllers today is that although they provide a lot of flexibility arround how traffic is routed through the devices they control, harnessing that flexibility. This can be alleviated by integrating information derived from machine learning of traffic flows into the controller logic, information which can help network administrators to simplify controller logic and configuration since they do not have to explicitly specify the logic required to match a certain type of traffic. This type of information can also shield the controller logic from changes in the underlying traffic patterns as well as have the network controller adapt to new unseen patterns.

Integrating information about the traffic patterns observed in the network only resolves half of the problem though since network administrators will need to decide what sort of actions to undertake in the end on that particular traffic and this type of decision is usually based on a much broader information context. This is not to say that learning cannot be used for that part as well but for the time being we concentrate on the first part of obtaining the traffic patterns, proving important results.

5. CONCLUSION

In this work we explored new ways to classify network traffic with the aid of machine learning techniques. We proposed a new approach to classify traffic based on statistical properties of the 5-dimension tuple unidirectional and bidirectional flows using:

- duration of the flow;
- total length of the flow;
- number of packets sent;
- minimum/maximum/average/standard deviation of packets length and inter-packet arrival times;
- first ten senders and some of other properties of the first ten packets in the flow.

We first grouped the flows extracted into a k number of clusters. k is an input parameter for the K-Means algorithm we used that was varied in Section 4, and we found that k should be as close as possible to the actual number of applications that are subject to the analyse. Having 14 protocols in our testing data, for the case when $k = 15$ we obtained accuracy of 85% for unidirectional flows and 86% for bidirectional flows. Another finding of this work is that the clustering operation performs better with bidirectional flows, because there are more statistical properties analysed and few flows to examine in total. Next, we used the output of the K-Means clusterization as input for a supervised learning, decision tree based algorithm, C4.5. Having 50%, 70%, and 90% of the input used for training and the other portion for testing, over 90% of the flows were classified correctly.

All in all, traffic clusterization of network flows performs well for well-chosen k values, and this findings could lead to supervised learning engines capable to deliver results able to accurately recognize and classify new and unknown traffic. Bidirectional flows proved to have better results than the unidirectional counterparts.

ACKNOWLEDGMENT

The research presented in this paper is supported by national project MobiWay, Project PN-II-PT-PCCA-2013-4-0321. The work was also supported by COST Action IC1303: Algorithms, Architectures and Platforms for Enhanced Living Environments (AAPELE).

REFERENCES

1. Feldmann A, Gilbert AC, Willinger W, Kurtz TG. The changing nature of network traffic: Scaling phenomena. *ACM SIGCOMM Computer Communication Review* 1998; **28**(2):5–29.
2. Livadas C, Walsh R, Lapsley D, Strayer WT. Usilng machine learning technliques to identify botnet traffic. *Local Computer Networks, Proceedings 2006 31st IEEE Conference on*, IEEE, 2006; 967–974.
3. Nguyen TT, Armitage G. A survey of techniques for internet traffic classification using machine learning. *Communications Surveys & Tutorials, IEEE* 2008; **10**(4):56–76.
4. Zander S, Nguyen T, Armitage G. Automated traffic classification and application identification using machine learning. *Local Computer Networks, 2005. 30th Anniversary. The IEEE Conference on*, IEEE, 2005; 250–257.
5. Kumar S, Turner J, Williams J. Advanced algorithms for fast and scalable deep packet inspection. *Proceedings of the 2006 ACM/IEEE symposium on Architecture for networking and communications systems*, ACM, 2006; 81–92.
6. Kumar S, Dharmapurikar S, Yu F, Crowley P, Turner J. Algorithms to accelerate multiple regular expressions matching for deep packet inspection. *ACM SIGCOMM Computer Communication Review* 2006; **36**(4):339–350.
7. Kandula S, Sengupta S, Greenberg A, Patel P, Chaiken R. The nature of data center traffic: measurements & analysis. *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, ACM, 2009; 202–208.
8. Moore AW, Zuev D. Internet traffic classification using bayesian analysis techniques. *ACM SIGMETRICS Performance Evaluation Review*, vol. 33, ACM, 2005; 50–60.
9. Miller B, Huang L, Joseph AD, Tygar JD. I know why you went to the clinic: Risks and realization of https traffic analysis. *Privacy Enhancing Technologies*, Springer, 2014; 143–163.
10. Jain AK. Data clustering: 50 years beyond k-means. *Pattern recognition letters* 2010; **31**(8):651–666.
11. Kanungo T, Mount DM, Netanyahu NS, Piatko CD, Silverman R, Wu AY. An efficient k-means clustering algorithm: Analysis and implementation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 2002; **24**(7):881–892.
12. Pelleg D, Moore A. Accelerating exact k-means algorithms with geometric reasoning. *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 1999; 277–281.
13. Pelleg D, Moore AW, *et al.*. X-means: Extending k-means with efficient estimation of the number of clusters. *ICML*, 2000; 727–734.
14. Hamerly G, Drake J. Accelerating lloyds algorithm for k-means clustering. *Partitional Clustering Algorithms*. Springer, 2015; 41–78.
15. Arthur D, Vassilvitskii S. k-means++: The advantages of careful seeding. *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, Society for Industrial and Applied Mathematics, 2007; 1027–1035.
16. Bouckaert RR, Frank E, Hall MA, Holmes G, Pfahringer B, Reutemann P, Witten IH. Weka—experiences with a java open-source project. *The Journal of Machine Learning Research* 2010; **11**:2533–2541.
17. Do CB, Batzoglou S. What is the expectation maximization algorithm? *Nature biotechnology* 2008; **26**(8):897–900.
18. Wang Y, Xiang Y, Zhang J. Network traffic clustering using random forest proximities. *Communications (ICC), 2013 IEEE International Conference on*, IEEE, 2013; 2058–2062.
19. McGregor A, Hall M, Lorier P, Brunskill J. Flow clustering using machine learning techniques. *Passive and Active Network Measurement*. Springer, 2004; 205–214.
20. Breiman L. Random forests. *Machine learning* 2001; **45**(1):5–32.
21. Wang Y, Yu SZ. Supervised learning real-time traffic classifiers. *Journal of Networks* 2009; **4**(7):622–629.
22. Li W, Canini M, Moore AW, Bolla R. Efficient application identification and the temporal and spatial stability of classification schema. *Computer Networks* 2009; **53**(6):790–809.
23. Williams N, Zander S, Armitage G. A preliminary performance comparison of five machine learning algorithms for practical ip traffic flow classification. *ACM SIGCOMM Computer Communication Review* 2006; **36**(5):5–16.
24. Dainotti A, Pescape A, Claffy KC. Issues and future directions in traffic classification. *Network, IEEE* 2012; **26**(1):35–40.
25. Bernaille L, Teixeira R, Akodkenou I, Soule A, Salamatian K. Traffic classification on the fly. *ACM SIGCOMM Computer Communication Review* 2006; **36**(2):23–26.
26. Duffield NG, Lewis JT, O'Connell N, Russell R, Toomey F. Entropy of atm traffic streams: a tool for estimating qos parameters. *Selected Areas in Communications, IEEE Journal on* 1995; **13**(6):981–990.