K-dimensional trees for continuous traffic classification

Valentín Carela-Español[1], Pere Barlet-Ros[1], Marc Solé-Simó[1], Alberto Dainotti[2], Walter de Donato[2], and Antonio Pescapé[2]

[1] Department of Computer Architecture, Universitat Politècnica de Catalunya (UPC)
vcarela,pbarlet,msole@ac.upc.edu
[2] Department of Computer Engineering and Systems, Universitá di Napoli Federico II
alberto,walter.dedonato,pescape@unina.it

Abstract. The network measurement community has proposed multiple machine learning (ML) methods for traffic classification during the last years. Although several research works have reported accuracies over 90%, most network operators still use either obsolete (e.g., port-based) or extremely expensive (e.g., pattern matching) methods for traffic classification. We argue that one of the barriers to the real deployment of ML-based methods is their time-consuming training phase. In this paper, we revisit the viability of using the Nearest Neighbor technique for traffic classification. We present an efficient implementation of this well-known technique based on multiple K-dimensional trees, which is characterized by short training times and high classification speed.This allows us not only to run the classifier online but also to continuously retrain it, without requiring human intervention, as the training data become obsolete. The proposed solution achieves very promising accuracy ($> 95\%$) while looking just at the size of the very first packets of a flow. We present an implementation of this method based on the TIE classification engine as a feasible and simple solution for network operators.

1 Introduction

Gaining information about the applications that generate traffic in an operational network is much more than mere curiosity for network operators. Traffic engineering, capacity planning, traffic management or even usage-based pricing are some examples of network management tasks for which this knowledge is extremely important. Although this problem is still far from a definitive solution, the networking research community has proposed several machine learning (ML) techniques for traffic classification that can achieve very promising results in terms of accuracy. However, in practice, most network operators still use either obsolete (e.g., port-based) or unpractical (e.g., pattern matching) methods for traffic identification and classification. One of the reasons that explains this slow adoption by network operators is the time-consuming training phase involving most ML-based methods, which often requires human supervision and manual inspection of network traffic flows.

In this paper, we revisit the viability of using the well-known Nearest Neighbor (NN) machine learning technique for traffic classification. As we will discuss throughout the paper, this method has a large number of features that make it very appealing for traffic classification. However, it is often discarded given its poor classification speed [11, 15]. In order to address this practical problem, we present an efficient implementation of the NN search algorithm based on a K-dimensional tree structure that allows us not only to classify network traffic online with high accuracy, but also to retrain the classifier on-the-fly with minimum overhead, thus lowering the barriers that hinder the general adoption of ML-based methods by network operators.

Our K-dimensional tree implementation only requires information about the length of the very first packets of a flow. This solution provides network operators with the interesting feature of *early classification* [2, 3]. That is, it allows them to rapidly classify a flow without having to wait until its end, which is a requirement of most previous traffic classification methods [7, 12, 16]. In order to further increase the accuracy of the method along with its classification speed, we combine the information about the packet sizes with the relevant data still provided by the port numbers [11].

We present an actual implementation of the method based on the Traffic Identification Engine (TIE) [5]. TIE is a community-oriented tool for traffic classification that allows multiple classifiers (implemented as plugins) to run concurrently and produce a combined classification result.

Given the low overhead imposed by the training phase of the method and the plugins already provided by TIE to set the ground truth (e.g., L7 plugin), the implementation has the unique feature of continuous training. This feature allows the system to automatically retrain itself as the training data becomes obsolete. We hope that the large advantages of the method (i.e., accuracy ($>$ 95%), classification speed, early classification and continuous training) can give an incentive to network operators to progressively adopt new and more accurate ML-based methods for traffic classification.

The remainder of this paper is organized as follows. Section II reviews the related work. Section III describes the ML-based method based on TIE. Section IV analyzes the performance of the method and presents preliminary results of its continuous training feature. Finally, Section V concludes the paper and outlines our future work.

2 Related Work

Traffic classification is a classical research area in network monitoring and several previous works have proposed different solutions to the problem. This section briefly reviews the progress in this research field, particularly focusing on those works that used the Nearest Neighbor algorithm for traffic classification.

Originally, the most common and simplest technique to identify network applications was based on the port numbers (e.g., those registered by the IANA [9]). This solution was very efficient and accurate with traditional applications. How-

ever, the arrival of new applications (e.g., P2P) that do not use a pre-defined set of ports or even use registered ones from other applications made this solution unreliable to classify current Internet traffic.

Deep packet inspection (DPI) constituted the first serious alternative to the well-known ports technique. DPI methods are based on searching for typical signatures of each application in the packet payloads. Although these techniques can potentially be very accurate, the high resource requirements of pattern matching algorithms and their limitations in the presence of encrypted traffic make their use incompatible with the continuously growing amount of data in current networks.

Machine learning techniques (ML) were later proposed as a promising solution to the well-known limitations of port- and DPI-based techniques. ML methods extract knowledge of the characteristic features of the traffic generated by each application from a training set. This knowledge is then used to build a classification model. We refer the interested reader to [13], where an extensive comparative study of existing ML methods for traffic classification is presented.

Among the different ML-based techniques existing in literature, the NN method rapidly became one of the most popular alternatives due to its simplicity and high accuracy. In general, given an instance p, the NN algorithm finds the nearest instance (usually using the Euclidean distance) from a training set of examples. NN is usually generalized to K-NN where K refers to the number of nearest neighbors to take into account.

The NN method for traffic classification was firstly proposed in [14], where a comparison of the NN technique with the Linear Discriminant Analysis method was presented. They showed that NN was able to classify, among 7 different classes of traffic, with an error rate below 10%.

However, the most interesting conclusions about the NN algorithm are found in the works from Williams et al. [15] and Kim et al. [11]. Both works compared different ML methods and showed the pros and cons of the NN algorithm for traffic classification. In summary, NN was shown to be one of the most accurate ML methods, with the additional feature of requiring zero time to build the classification model. However, NN was the ML-based algorithm with the worst results in terms of classification speed. This is the reason why NN is often discarded for online classification.

The efficient implementation of the NN algorithm presented in this paper is based instead on the K-dimensional tree, which solves its problems in terms of classification speed, while keeping very high accuracy. Another important feature of the method is its ability to early classify the network traffic. This idea is exported from the work from Bernaille et al. [2, 3]. This early classification feature allows the method to classify the network traffic by just using the first packets of each flow. Bernaille et al. compared three different unsupervised ML methods (K-Means, GMM and HMM), while in this work we apply this idea to a supervised ML method (NN).

As ML-based methods for traffic classification become more popular, new techniques appear in order to evade classification. These techniques, such as

protocol obfuscation, modify the value of the features commonly used by the traffic classification methods (e.g., by simulating the behavior of other applications or padding packets). Several alternative techniques have been also proposed to avoid some of these limitations. BLINC [10] is arguably the most well-known exponent of this alternative branch. Most of these methods base their identification in the behavior of the end-hosts and, therefore, their accuracy is strongly dependent on the network viewpoint where the technique is deployed [11].

3 Methodology

This section describes the ML-based classification method based on multiple K-dimensional trees, together with its continuous training system. We also introduce TIE, the traffic classification system we use to implement our technique, and the modifications made to it in order to allow the method to continuously retrain itself.

3.1 Traffic Identification Engine

TIE [5] is a modular traffic classification engine developed by the Universitá di Napoli Federico II. This tool is designed to allow multiple classifiers (implemented as plugins) to run concurrently and produce a combined classification result. In this work, we implement the traffic classification method as a TIE plugin.

TIE is divided in independent modules that are in charge of the different classification tasks. The first module, *Packet Filter*, uses the Libpcap library to collect the network traffic. This module can also filter the packets according to BPF or user-level filters (e.g., skip the first n packets, check header integrity or discard packets in a time range). The second module, *Session Builder*, aggregates packets in flows (i.e., unidirectional flows identified by the classic 5-tuple), biflows (i.e., both directions of the traffic) or host sessions (aggregation of all the traffic of a host). The *Feature Extractor* module calculates the features needed by the classification plugins. There is a single module for feature extraction in order to avoid redundant calculations for different plugins. TIE provides a multi-classifier engine divided in a *Decision Combiner* module and a set of classification plugins. On the one hand, the *Decision Combiner* is in charge of calling several classification plugins when their features are available. On the other hand, this module merges the results obtained from the different classification plugins in a definitive classification result. In order to allow comparisons between different methods, the *Output* module provides the classification results from the *Classification Combiner* based on a set of applications and groups of applications defined by the user.

TIE supports three different operating modes. The *offline mode* generates the classification results at the end of the TIE execution. The *real-time mode* outputs the classification results as soon as possible, while the *cycling mode* is an hybrid mode that generates the information every n minutes.

3.2 KD-Tree plugin

In order to evaluate the traffic classification method, while providing a ready-to-use tool for network operators, we implement the K-dimensional tree technique as a TIE plugin. Before describing the details of this new plugin, we introduce the K-dimensional tree technique. In particular, we focus on the major differences with the original NN search algorithm.

The K-dimensional tree is a data structure to efficiently implement the Nearest Neighbor search algorithm. It represents a set of N points in K-dimensional spaces as described by Friedman et al. [8] and Bentley [1]. In the naive NN technique the set of points is represented as a set of vectors where each position of a vector represents a coordinate from a point (i.e., feature). Besides these data, the K-dimensional tree implementation also creates a binary tree that recursively take the median point of the set of points, leaving half of points in each side.

The original NN algorithm searches iteratively the nearest point i, from a set of points E, to a point p. In order to find the i point, it computes, for each point in E, the distance (e.g., Euclidean or Manhattan distance) to the point p. Likewise, if we are performing a K-NN search, the algorithm looks for the K i points nearest to the point p. This search has O(N) time complexity and becomes unpractical with the amount of traffic found in current networks.

On the contrary, the search in a K-dimensional tree allows to find in average the nearest point in O(log N), with the additional cost of spending once O(N log N) building the binary tree. Besides this notable improvement, the structure also supports approximate searches, which can substantially improve the classification time at the cost of producing a very small error.

The K-dimensional tree plugin that we implement in TIE is a combination of the K-dimensional tree implementation provided by the C++ ANN library and a structure to represent the relevant information still provided by the port numbers. In particular, we create an independent K-dimensional tree for each *relevant* port. We refer as *relevant* ports as those that generate more traffic. Although the list of *relevant* ports can be computed automatically, we also provide the user with the option of manually configuring this list. Another configuration parameter is the approximation value, which allows the method to improve its classification speed by performing an approximate NN search. In the evaluation, we set this parameter to 0, which means that this approximation feature is not used. However, higher values of this parameter could substantially improve the classification time in critical scenarios, while still obtaining a reasonable accuracy.

Unlike in the original NN algorithm, the proposed method requires a lightweight training phase to build the K-dimensional tree structure. Before building the data structure, a sanitation process is performed on the training data. This procedure removes the instances labeled as unknown from the training dataset assuming that they have similar characteristics to other known flows. This assumption is similar to that of ML clustering methods, where unlabeled instances are classified according to their proximity in the feature space to those that are known. The sanitation process also removes repeated or indistinguishable instances.

The traffic features used by our plugin are the destination port number and the length of the first n packets of a flow (without considering the TCP handshake). By using only the first n packets, the plugin can classify the flows very fast, providing the network operator with the possibility of quickly reacting to the classification results. In order to accurately classify short flows, the training phase also accepts flows with less than n packets by filling the empty positions with null coordinates.

3.3 Continuous training system

In this section, we show the interaction of our *KD-Tree* plugin with the rest of the TIE architecture, and describe the modifications done in TIE to allow our plugin to continuously retrain itself.

Figure 1 shows the data flow of our continuous training system based on TIE. The first three modules are used without any modification as found in the original version of TIE. Besides the implementation of the new *KD-Tree* plugin, we significantly modified the *Decision Combiner* module and the *L7* plugin.

Our continuous training system follows the original TIE operation mode most part of the time. Every packet is aggregated in bidirectional flows while its features are calculated. When the traffic features of a flow (i.e., first n packet sizes) are available or upon its expiration, the flow is classified by the *KD-Tree* plugin. Although the method was tested with bidirectional flows, the current implementation also supports the classification of unidirectional flows.

In order to automatically retrain our plugin, as the training data becomes obsolete, we need a technique to set the base-truth. TIE already provides the *L7* plugin, which implements a DPI technique originally used by TIE for validation purposes. We modified the implementation of this plugin to continuously produce training data (which includes flow labels - that is, the base-truth - obtained by L7) for future trainings. While every flow is sent to the *KD-Tree* plugin through the main path, the *Decision Combiner* module applies flow sampling to the traffic, which is sent through a secondary path to the *L7* plugin. This secondary path is used to *(i)* set the base truth for the continuous training system, *(ii)* continuously check the accuracy of the *KD-Tree* plugin by comparing its output with that of *L7*, and *(iii)* keep the required computational power low by using flow sampling (performing DPI on every single flow will significantly decrease the performance of TIE).

The *Decision Combiner* module is also in charge of automatically triggering the training of the *KD-Tree* plugin according to three different events that can be configured by the user: after p packets, after s seconds, or if the accuracy of the plugin compared to the *L7* output is below a certain threshold t. The flows classified by the *L7* plugin, together with their features (i.e., destination port, n packet sizes, L7 label), are placed in a queue. This queue keeps the last f classified flows or the flows classified during the last s seconds.

The training module of the *KD-Tree* plugin is executed in a separate thread. This way, the *KD-Tree* plugin can continuously classify the incoming flows without interruption, while it is periodically updated. The training module builds a

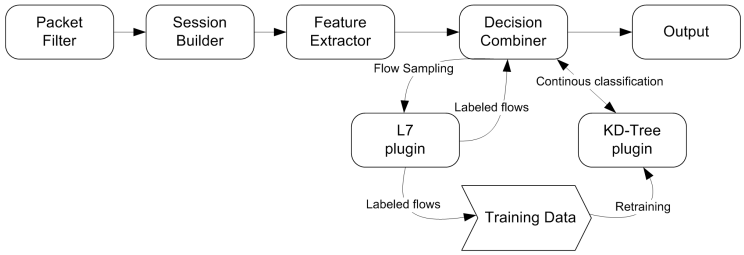


Fig. 1. Diagram of the Continuous Training Traffic Classification system based on TIE

completely new multi K-dimensional tree model using the information available in the queue. We plan as future work to study the alternative solution of incrementally updating the old model with the new information, instead of creating a new model from scratch. In addition, it is possible to automatically update the list of *relevant* ports by using the training data as a reference.

4 Results

This section presents a performance evaluation of the proposed technique. First, Subsection 4.1 describes the dataset used in the evaluation. Subsection 4.2 compares the original Nearest Neighbor algorithm with the K-dimensional tree implementation. Subsection 4.3 presents a performance evaluation of the proposed plugin described in Subsection 3.2 and, evaluates different aspects of the technique as the *relevant* ports or the number of packet sizes used for the classification. Finally, Subsection 4.4 presents a preliminary study of the impact of the continuous training system in the traffic classification.

4.1 Evaluation Datasets

The dataset used in our performance evaluation consists of 8 full-payload traces collected at the Gigabit access link of the Universitat Politècnica de Catalunya (UPC), which connects about 25 faculties and 40 departments (geographically distributed in 10 campuses) to the Internet through the Spanish Research and Education network (RedIRIS). This link provides Internet access to about 50000 users. The traces were collected at different days and hours trying to cover as much diverse traffic from different applications as possible. Due to privacy issues, we are not able to publish our traces. However, we made our traces accessible using the CoMo-UPC model presented in [4].

Table 1 presents the details of the traces used in the evaluation. In order to evaluate the proposed method, we used the first seven traces. Among those traces, we selected a single trace (*UPC-II*) as training dataset, which is the trace that contains the highest diversity in terms of instances from different applications. We limited our training set to one trace in order to leave a meaningful

Table 1. Characteristics of the traffic traces in our dataset

Name	Date	Day	Start Time	Duration	Packets	Bytes	Valid Flows	Avg. Util
UPC-I	11-12-08	Thu	10:00	15 min	95 M	53 G	1936 K	482 Mbps
UPC-II	11-12-08	Thu	12:00	15 min	114 M	63 G	2047 K	573 Mbps
UP-III	12-12-08	Fri	01:00	15 min	69 M	38 G	1419 K	345 Mbps
UPC-IV	12-12-08	Fri	16:00	15 min	102 M	55 G	2176 K	500 Mbps
UPC-V	14-12-08	Sun	00:00	15 min	53 M	29 G	1346 K	263 Mbps
UPC-VI	21-12-08	Sun	12:00	1 h	175 M	133 G	3793 K	302 Mbps
UPC-VII	22-12-08	Mon	12:30	1 h	345 M	256 G	6684 K	582 Mbps
UPC-VIII	10-03-09	Tue	03:00	1 h	114 M	78 G	3711 K	177 Mbps

number of traces for the evaluation that are not used to build the classification model. Therefore, the remaining traces were used as the validation dataset. The last trace, *UPC-VIII*, was recorded with a difference in time of four months with the trace *UPC-II*. Given this time difference, we used this trace to perform a preliminary experiment to evaluate the gain provided by our continuous training solution.

4.2 Nearest Neighbor vs K-dimensional Tree

Section 3.2 already discussed the main advantages of the K-dimensional tree technique compared to the original Nearest Neighbor algorithm. In order to present numerical results showing this gain, we perform a comparison between both methods. We evaluate the method presented in this paper with the original NN search implemented for validation purposes by the ANN library. Given that the ANN library implements both methods in the same structure we calculated the theoretical minimum memory resources necessary for the naive NN technique (i.e., *# unique examples * # packet sizes * 4 bytes (C++ integer)*). We tested both methods with the trace *UPC-II* (i.e., ≈500.000 flows after the sanitation process) using a 3GHz machine with 4GB of RAM. It is important to note that, since we are performing an offline evaluation, we do not approximate the NN search in the NN original algorithm or in the K-dimensional tree technique. For this reason, the accuracy of both methods is the same.

Table 2 summarizes the improvements obtained with the combination of the K-dimensional tree technique with the information from the port numbers. Results are shown in terms of classifications per second depending on the number of packets needed for the classification and the list of *relevant* ports. There are three possible lists of *relevant* ports. The *unique* list, where there are no *relevant* ports and all the instances belong to the same K-dimensional tree or NN structure. The *selected* list, which is composed by the set of ports that contains most traffic from the *UPC-II* trace (i.e., ports that receive more than 0.05% of the traffic (69 ports in the *UPC-II* trace)). We finally refer to *all* as the

Table 2. Speed Comparison (flows/s): Nearest Neighbor vs K-Dimensional Tree

Packet Size	Naive Nearest Neighbor			K-Dimensional Tree		
	Unique	Selected Ports	All Ports	Unique	Selected Ports	All Ports
1	45578	104167	185874	423729	328947	276243
5	540	2392	4333	58617	77280	159744
7	194	1007	1450	22095	34674	122249
10	111	538	796	1928	4698	48828

Table 3. Memory Comparison: Nearest Neighbor vs K-Dimensional Tree

Packet Size	Naive	K-Dimensional Tree		
	Nearest Neighbor	Unique	Selected Ports	All Ports
1	2.15 MB	40.65 MB	40.69 MB	40.72 MB
5	10.75 MB	52.44 MB	52.63 MB	53.04 MB
7	15.04 MB	56.00 MB	56.22 MB	57.39 MB
10	21.49 MB	68.29 MB	68.56 MB	70.50 MB

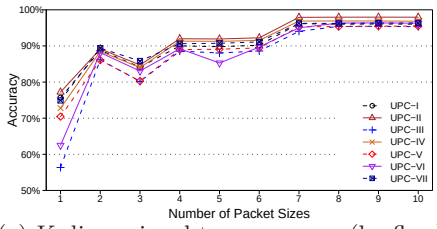
list where all ports found in the *UPC-II* trace are considered as *relevant*. The first column corresponds to the original NN presented in previous works [11, 14, 15], where all the information is maintained in a single structure. When only one packet is required, the proposed method is ten times faster than the original NN. However, the speed of the original method dramatically decreases when the number of packets required increases, becoming even a hundred times slower than the K-dimensional tree technique. In almost all the situations, the introduction of the list of *relevant* ports substantially increases the classification speed in both methods.

Tables 3 and 4 show the extremely low price that the K-dimensional tree technique pays for a notable improvement in classification speed. The results show that the memory resources required by the method, although being higher than the naive NN technique, are few. The memory used in the K-dimensional tree is almost independent from the *relevant* ports parameter and barely affected by the number of packet sizes. Regarding time, we believe that the trade-off of the training phase is well compensated by the ability to use the method as an online classifier. In the worst case, the method only takes about 20 seconds for the building phase.

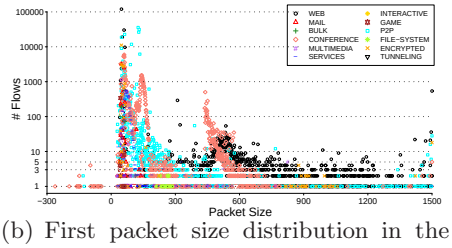
Since both methods output the same classification results, the data presented in this subsection show that the combination of the *relevant* ports and the K-dimensional tree technique significantly improves the original NN search with the only drawback of a (very fast) training phase. This improvement allows us to use this method as an efficient online traffic classifier.

Table 4. Building Time Comparative: Nearest Neighbor vs K-Dimensional Tree

Packet Size	Naive Nearest Neighbor	K-Dimensional Tree		
		Unique	Selected Ports	All Ports
1	0 s	13.01 s	12.72 s	12.52 s
5	0 s	16.45 s	16.73 s	15.62 s
7	0 s	17.34 s	16.74 s	16.07 s
10	0 s	19.81 s	19.59 s	18.82 s



(a) K-dimensional tree accuracy (by flow) without *relevant* ports support



(b) First packet size distribution in the training trace *UPC-II*

Fig. 2. K-dimensional tree evaluation without the support of the *relevant* ports

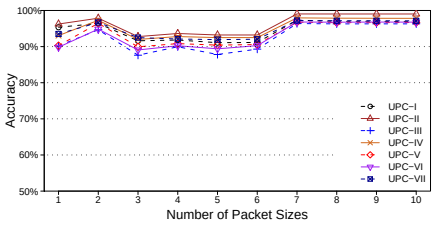
4.3 K-dimensional tree plugin evaluation

In this section we study the accuracy of the method depending on the different parameters of the *KD-Tree* plugin. Figure 2(a) presents the accuracy according to the number of packet sizes for the different traces of the dataset. In this case, no information from the *relevant* ports is taken into account producing a single K-dimensional tree. With this variation, using only the first two packets, we achieve an accuracy of almost 90%. The accuracy increases with the number of packet sizes until a stable accuracy $> 95\%$ is reached with seven packet sizes.

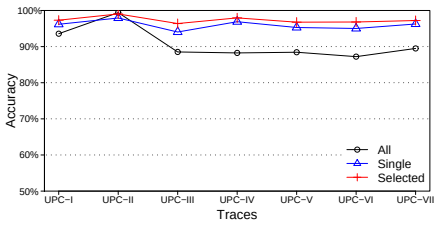
In order to show the impact of using the list of *relevant* ports in the classification, in Figure 2(b) we show the distribution of the first packet sizes for the training trace *UPC-II*. Although there are some portions of the distribution dominated by a group of applications, most of the applications have their first packet sizes between the 0 and the 300 bytes ticks. This collision explains the poor accuracy presented in the previous figure with only one packet.

The second parameter of the method, the *relevant* ports, besides improving the classification speed appears to alleviate that situation. Figure 3(a) presents the accuracy of the method by number of packets using the set of *relevant* ports that contains most of the traffic in *UPC-II*. With the help of the *relevant* ports, the method achieves an accuracy $> 90\%$ using only the first packet size and achieving a stable accuracy of 97% with seven packets.

Figure 3(b) presents the accuracy of the method depending on the set of *relevant* ports with seven packet sizes. We choose seven because as it can be seen



(a) K-dimensional tree accuracy (by flow) with *relevant* ports support



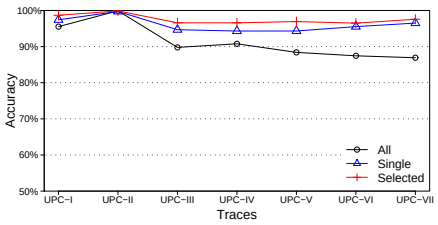
(b) K-dimensional tree accuracy (by flow, n=7) by set of *relevant* ports

Fig. 3. K-dimensional tree evaluation with the support of the *relevant* ports

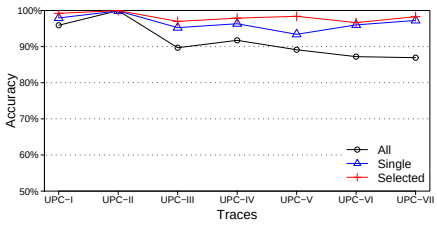
in Figures 2(a) and 3(a), increasing the number of packet sizes beyond seven does not improve its accuracy but decrease, its classification speed. Using all the ports of the training trace *UPC-II*, the method achieves the highest accuracy with the same trace. However, with the rest of the traces the accuracy substantially decreases but being always higher than 85%. The reason of this decrease is that using all the ports as *relevant* ports is very dependent to the scenario and could present classification inaccuracies with new instances belonging to ports not represented in the training data. Furthermore, the classification accuracy also decreases because it produces fragmentation in the classification model for those applications that use multiple or dynamic ports (i.e., their information is spread among different K-dimensional trees). However, the figure shows that using a set of *relevant* ports - in our case the ports that receive more than 0.05% of the traffic - besides increasing the classification speed also improves accuracy.

Erman et al. pointed out in [6] a common situation found among the ML techniques: the accuracy when measured by flows is much higher than when measured by bytes or packets. This usually happens because some elephant-flows are not correctly classified. Figures 4(a) and 4(b) present the classification results of the method considering also the accuracy by bytes and packets. They show that, unlike other ML solutions, the method is able to keep high accuracy values even with such metrics.This is because the method is very accurate with the group of applications *P2P* and *WEB*, which represent in terms of bytes most of the traffic in our traces.

Finally, we also study the accuracy of the method broken down by application group. In our evaluation we use the same application groups as in TIE. Figure 5 shows that the method is able to classify with excellent accuracy the most popular groups of applications. However, the accuracy of the applications groups that are not very common substantially decreases. These accuracies have a very low impact on the final accuracy of the method given that the representation of these groups in the used traces is almost negligible. A possible solution to improve the accuracy for these groups of applications could be the addition of artificial instances of these groups in the training data. Another potential problem is the disguised use of ports by some applications. Although we do not



(a) K-dimensional tree accuracy (by packet, n=7) by set of *relevant* ports



(b) K-dimensional tree accuracy (by byte, n=7) by set of *relevant* ports

Fig. 4. K-dimensional tree evaluation with the support of the *relevant* ports

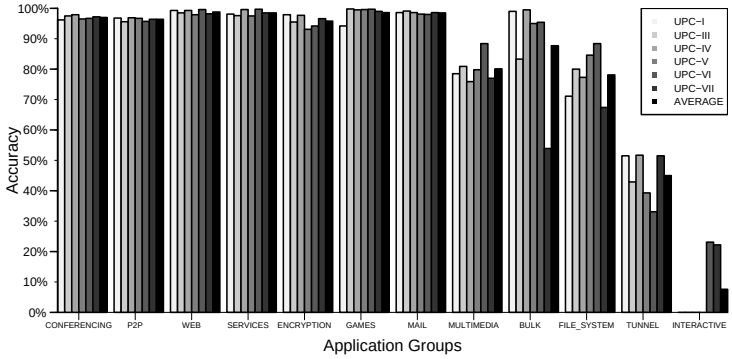


Fig. 5. Accuracy by application group (n=7 and *selected* list of ports as parameters)

have evaluated this impact in detail, the results show that currently we can still achieve an additional gain in accuracy by considering the port numbers. We have also checked the accuracy by application group with a single K-dimensional tree and we found that it was always below the results shown in Figure 5. We omit the figure in the interest of brevity.

In conclusion, we presented a set of results showing how the K-dimensional tree technique, combined with the still useful information provided by the ports, improves almost all the aspects of previous methods based in the NN search. With the unique drawback of a short training phase, the method is able to perform online classification with very high accuracy, $> 90\%$ with only one packet or $> 97\%$ with seven packets.

4.4 Continuous training system evaluation

This section presents a preliminary study of the impact of our continuous training traffic classifier. Due to lack of traces comprising a very long period of time and because of the intrinsic difficulties in processing such large traces, we simulate a scenario in which the features of the traffic evolve by concatenating the *UPC-II*

Table 5. Evaluation of the Continuous Training system by training trace and set of *relevant* ports

Training Trace	UPC-II		First 15 min. UPC-VIII	
Relevant Port List	UPC-II	UPC-VIII	UPC-II	UPC-VIII
Accuracy	84.20 %	76.10 %	98.17 %	98.33 %

and *UPC-VIII* traces. The trace *UPC-VIII*, besides belonging to a difference day-time, was recorded four months later than *UPC-II*, this suggests a different traffic mix with different properties. Using seven as the fixed number of packets sizes, the results in Table 5 confirm our intuition. On one hand, using the trace *UPC-II* as training data to classify the trace *UPC-VIII* we obtain an accuracy of almost 85%. On the other hand, after detecting such decrease in accuracy and retraining the system, we obtain and impressive accuracy of 98,17%. This result shows the importance of the continuous training feature to maintain a high classification accuracy. Since this preliminary study was performed with traffic traces, instead of a live traffic stream, we decided to use the first fifteen minutes of the *UPC-VIII* trace as the queue length parameter (s) of the retraining process.

The results of a second experiment are also presented in Table 5. Instead of retraining the system with a new training data we study if the modification of the list of *relevant* ports is enough to obtain the original accuracy. The results show that this solution does not bring any improvement when applied alone. However the optimum solution is obtained when both the training data and the list of *relevant* ports are updated and the system is then retrained.

5 Conclusions and future work

In this paper, we revisited the viability of using the Nearest Neighbor algorithm (NN) for online traffic classification, which has been often discarded in previous studies due to its poor classification speed. In order to address this well-known limitation, we presented an efficient implementation of the NN algorithm based on a K-dimensional tree data structure, which can be used for online traffic classification with high accuracy and low overhead. In addition, we combined this technique with the relevant information still provided by the port numbers, which further increases its classification speed and accuracy.

Our results show that the method can achieve very high accuracy ($> 90\%$) by looking only at the first packet of a flow. When the number of analyzed packets is increased to seven, the accuracy of the method increases beyond 95%. This early classification feature is very important, since it allows network operators to quickly react to the classification results.

We presented an actual implementation of the traffic classification method based on the TIE classification engine. The main novelty of the implementation is its continuous training feature, which allows the system to be automatically

retrained by itself as the training data becomes obsolete. Our preliminary evaluation of this unique feature presents very encouraging results.

As future work, we plan to perform a more extensive performance evaluation of our continuous training system with long-term executions in order to show the large advantages of maintaining the classification method continuously updated without requiring human supervision.

Acknowledgments

This paper was done under the framework of the COST Action IC0703 *"Data Traffic Monitoring and Analysis (TMA)"* and with the support of the Comissionat per a Universitats i Recerca del DIUE from the Generalitat de Catalunya. The authors thank UPCnet for the traffic traces provided for this study and the anonymous reviewers for their useful comments.

References

1. Bentley, J.L.: K-d trees for semidynamic point sets pp. 187–197 (1990)
2. Bernaille, L., Teixeira, R., Salamatian, K.: Early application identification. In: Proc. of ACM CoNEXT (2006)
3. Bernaille, L., et al.: Traffic classification on the fly. ACM SIGCOMM Comput. Commun. Rev. 36(2) (2006)
4. CoMo-UPC data sharing model: http://monitoring.ccaba.upc.edu/como-upc/
5. Dainotti, A., et al.: TIE: a community-oriented traffic classification platform. In: Proceedings of the First International Workshop on Traffic Monitoring and Analysis. p. 74 (2009)
6. Erman, J., Mahanti, A., Arlitt, M.: Byte me: a case for byte accuracy in traffic classification. In: Proc. of ACM SIGMETRICS MineNet (2007)
7. Erman, J., et al.: Identifying and discriminating between web and peer-to-peer traffic in the network core. In: Proc. of WWW Conf. (2007)
8. Friedman, J.H., Bentley, J.L., Finkel, R.A.: An algorithm for finding best matches in logarithmic expected time. ACM Trans. Math. Softw. 3(3), 209–226 (1977)
9. Internet Assigned Numbers Authority (IANA): http://www.iana.org/assignments/port-numbers, as of August 12, 2008
10. Karagiannis, T., Papagiannaki, K., Faloutsos, M.: BLINC: multilevel traffic classification in the dark. In: Proc. of ACM SIGCOMM (2005)
11. Kim, H., et al.: Internet traffic classification demystified: myths, caveats, and the best practices. In: Proc. of ACM CoNEXT (2008)
12. Moore, A., Zuev, D.: Internet traffic classification using bayesian analysis techniques. In: Proc. of ACM SIGMETRICS (2005)
13. Nguyen, T., Armitage, G.: A survey of techniques for internet traffic classification using machine learning. IEEE Communications Surveys and Tutorials 10(4) (2008)
14. Roughan, M., et al.: Class-of-service mapping for qos: a statistical signature-based approach to ip traffic classification. In: Proc. of ACM SIGCOMM IMC (2004)
15. Williams, N., Zander, S., Armitage, G.: Evaluating machine learning algorithms for automated network application identification. CAIA Tech. Rep. (2006)
16. Zander, S., Nguyen, T., Armitage, G.: Automated traffic classification and application identification using machine learning. In: Proc. of IEEE LCN Conf. (2005)