Efficient Methods for Early Protocol Identification

Béla Hullár, Sándor Laki, András György

Abstract—**To manage and monitor their networks in a proper way, network operators are often interested in automatic methods that enable them to identify applications generating the traffic traveling through their networks as fast (i.e., from the first few packets) as possible. State-of-the-art packet-based traffic classification methods are either based on costly inspection of the payload of several packets in each flow or on basic flow statistics without taking into account the packet content. In this paper, we consider an intermediate approach of analyzing only the first few bytes of the first (or first few) packet(s) of each flow and propose automatic, machine-learning-based methods with very low computational complexity and memory footprint. The performance of these techniques are thoroughly analyzed, showing that outstanding early classification accuracy can be achieved on traffic traces generated by a diverse set of applications (including P2P TV and file sharing) in a laboratory environment as well as on a real-world data set collected in the network of a large European ISP.**

Index Terms—**Traffic classification, payload statistics, machine learning**

I. Introduction

THE assurance of quality of service in IP-based networks, especially in the Internet, is one of the most important challenges in today's network development and management. To ensure quality of service, network operators have to analyze the traffic traveling through their networks. This analysis can lead to more efficient resource allocation as well as can help planning the expansion and development of the network. This is the reason why traffic classification is in the forefront of research on next generation networks.

A decade ago, traffic classification was a very simple task since applications used their well defined port numbers assigned by IANA and the network operators just looked at the transport layer port number for identification. Since then the Internet has became the largest existing entertainment medium in the world. Many new applications have appeared from voice over IP (VoIP) and IP TV to peer-to-peer (P2P) file sharing systems that have a revolutionary effect on people's everyday life. Nowadays these applications are responsible for the majority of network traffic. To by-pass firewalls and routers, or to hide their presence, they use dynamic port numbers or well-known trusted ports (such as HTTP or SMTP ports), limiting the achievable performance of port-number-based classification.

Current state-of-the-art methods for traffic classification are based on the analysis of (i) packet payloads generated by the application; (ii) some network layer or transport layer features; (iii) host-behavior, or some mixtures of the above. Approach (i), called deep packet inspection (DPI) aims at identifying typical protocol patterns in the messages of different applications. These methods work reasonably well in practice, but their disadvantages are well-known: The performed pattern matching is resource-intensive, requiring both computational time for scanning packet payloads and memory for storing large databases of application (or protocol) specific signatures. Moreover, keeping the signature databases up-to-date is a cumbersome task usually requiring human expertise, since new network-based applications are born day-by-day and the current ones evolve constantly. To eliminate these drawbacks, statistical payload inspection methods have been proposed recently [1], [2] that learn relevant protocol patterns automatically from traffic traces (labeled with the corresponding applications).

In some situations packet content may not be available due to resource limitations or encryption. Then the less resource intensive approaches (ii) and (iii) may be applied. In the former, machine learning methods (such as decision trees, Bayesian classifiers, SVM, k-NN, etc.) are used to train classifiers to estimate the traffic type based on some features extracted from some network or transport layer data, such as the number of packets in the flow, the average packet size, the count of Push flags, inter-arrival time statistics, etc. In the last few years, numerous feature sets and machine learning algorithms were examined in the literature, see, for example, the thorough review [3] or the systematic comparison of existing methods in [4]. However, the efficiency of different feature sets highly depends on the generating applications as well as on the location where the flow data was captured, limiting the applicability of the results in a universal traffic classification tool [5]. Finally, approach (iii) is based on the analysis of host behavior instead of individual network connections. The basic idea

Manuscript received December 15, 2013; revised May 12, 2014. This work was supported in part by the grant EITKIC-12-1-2012-0001 of the National Development Agency, and the EU FP7 OpenLab project (Grant No.287581), the National Development Agency of Hungary from the Research and Technological Innovation Fund (KTIA-OTKA CNK 77782), the Alberta Innovates Technology Futures and NSERC. S. Laki also thanks the partial support of MAKOG foundation.

This work was presented in part at the IEEE International Conference on Communications (ICC 2011) Next Generation Networking and Internet Symposium, Kyoto, Japan, June 2011.

B. Hullár was with the Department of Physics of Complex Systems, Eötvös Loránd University, Budapest, Hungary. He is now with the Scientific IT Services, ETH Zürich, Switzerland (e-mail: behullar@ethz.ch).

S. Laki is with the Department of Information Systems, Eötvös Loránd University, Budapest, Hungary (e-mail: lakis@inf.elte.hu).

A. György is with the Department of Computing Science, University of Alberta, Edmonton, AB, Canada; during part of this work he was with the Machine Learning Research Group, Computer and Automation Research Institute of the Hungarian Academy of Sciences, Budapest, Hungary (e-mail: gyorgy@ualberta.ca).

behind this solution is that different applications or application groups have different communication habits, for example, a client application has few outgoing connections, a server has numerous incoming connections, while P2P applications have many connections in both directions, giving rise to methods analyzing the connection patterns of host interactions [6], [7] or utilizing the assumption that similar hosts tend to be related to one another [8].

An important observation is that classification algorithms analyzing packet content outperform others in case of unencrypted traffic, while the rest are much less resource intensive and, hence, much cheaper. Furthermore, identifying the application type as early as possible is very important from several aspects (from security to improved quality of service) allowing the operator much faster reactions. To this end, the first few bytes of each flow are used for making decisions in [2], while [9] performs classification from some simple descriptors of the first four packets of each flow.

In this paper we tackle the above problems, and, by extending and improving the work of [2], demonstrate that automated traffic classification based on the first few bytes of the first (or first few) packet(s) of a connection is possible with high accuracy. In addition, some of our approaches require far less computational and memory resources than usual DPIs with reasonably good performance, and no human expertise is needed in the development (training) of the methods. It is worth noting that [10] and [4] have also found that analyzing only the first 16 bytes of a P2P connection is usually sufficient for a well designed DPI to identify P2P traffic (except for Gnutella).

The performance of our traffic classification algorithms is measured on a diverse set of data recorded in our lab over LAN, WiFi and 3G links, and in the network of a large European ISP. In our laboratory experiments, similarly to, for example, [11], we recorded individual traffic traces of several applications (mostly concentrated on P2P protocols) over different network architectures. Although one might think that such an setup may result in artificial user behavior and, hence, traffic patterns, the content of the first few packets does not seem to be really effected. This shows another advantage of our early classification approach: the ease of obtaining training data for previously unseen applications and traffic classes, and network architectures. Besides our laboratory measurements, we have also verified our methodology on real traffic traces collected in the network of an ISP, considering a large number of different protocols.

The rest of the paper is organized as follows. In Section II we briefly overview the most relevant statistical payload-based traffic classification methods. Our proposed early classification algorithms are introduced in Section II-A, including a brief comparison of their computational and storage complexities. Section III describes our data collection methodology. A comprehensive performance analysis of our algorithms in laboratory environment is presented in Section IV, while in Section V we show how our approaches perform on real world traffic traces. The advantages and limitations of our approach are discussed in Section VI. Finally, conclusions are drawn and future work is outlined in Section VII.

II. Payload-based Early Classification Using Statistical Methods

Since the majority of network traffic is still unencrypted and the protocol-format-based identification works well in practice, it is worth exploiting the potential of the payload analysis with advanced techniques. A very promising method in the literature is the KISS algorithm [1] that uses χ^2-statistics as features computed from the first few bytes of the first several packets of each flow. The efficiency of this solution was demonstrated on different applications, such as P2PTV clients, where, by combining the payload-based classification with other techniques, based on, for example, host similarities, they managed to achieve 99% accuracy in flow classification. The main disadvantage of this approach is the large number of packets (approximately 80) required for the feature computation to achieve the above results (and, accordingly, the measurements are restricted to flows that generate more than 80 packets), and this is why this method is not applicable for early classification.

In [2] simple statistical methods are considered for traffic classification from only the first 64 bytes of the network flows. This work proposes three different solutions based on a stationary memoryless model, a first-order Markov model over the bytes and a graph based common substring method, respectively, to analyze the beginning of the payload data. These methods show promising results on the limited test set containing only a few basic protocols, such as HTTP, DNS or NTP.

In this paper we extend the latter approach in several ways: we apply more involved models, such as context trees or random forests, our algorithms focus on only the first few bytes of a few packets per flow, and are validated on more recent applications, such as P2PTV and file sharing systems as well. Furthermore, our algorithms are also verified on real traffic traces captured in the network of a large ISP.

A. Early classification algorithms

We examine the problem of classifying traffic flows from very limited information when only the first N bytes of the payload of the first n packets of each flow are taken into account, and we show that very good accuracy can be reached even for small values of N and n, making our approaches more valuable in practice. By combining the limited data obtained from the selected packets, each flow can be described by a feature vector of nN bytes, representing a fingerprint of the observed flow that is used to determine the generating application.

Our classification framework can be described as follows: A sample of flows generated from the traffic classes to be identified is collected and each flow is labeled with its true class (i.e., the application generating the flow). Then a classification method is trained based on the feature-vector–label pairs corresponding to the generated flows (called the training data). Finally, the trained classifier is used to estimate the traffic class from the feature vectors. Most of our methods (except for the random forrest classifier) build a stochastic model for each class in the training phase, and a flow is estimated to belong to the class whose model fits the flow data the best (in a maximum likelihood sense). This classification methodology was used in [2]), where a first-order Markov model is used. In this paper we use more refined models and estimation strategies, as described below.

In the following we briefly overview the applied classification techniques including Markov-model-based classification methods of different orders as well as random forests: For Markovian methods, we build a stochastic (Markov) model for each traffic class i during the training phase that provides an estimate $p_i(\mathbf{x})$ of the conditional probability that we observe a feature vector $\mathbf{x} = (x_1, \ldots, x_{nN})$ if the flow was generated from class i. Then a maximum likelihood decision is applied to select the most likely class for the given feature vector $\mathbf{x}$, that is, the protocol class is estimated as $\operatorname{argmax}_i p_i(\mathbf{x})$. To estimate the probability $p_i(\mathbf{x})$ we have considered and analyzed various models of different orders.

KT-ESTIMATOR: Our simplest approach is to fit a zero-order model to the feature vectors. That is, the bytes in the feature vectors are assumed to be independent with identical distributions (given the traffic class), and one needs to estimate the distribution of the bytes in the feature vectors. In this model it is natural to use the empirical byte frequencies to estimate the true distribution; however, this approach is not robust in assigning probabilities to symbols (bytes) not present in the training data. To alleviate this problem, we use the Krichevsky-Trofimov estimator [12]: For class i, let n_i denote the total number of bytes in the feature vectors belonging to the given class in the training data. Similarly, let $n_i(x)$ denote the total number of occurrences of byte x in the feature vectors of class i in the training set (it is obvious that $\sum_x n_i(x) = n_i$). Then, the probability of byte x for class i is estimated by

$$p_i(x) = (n_i(x) + 1/2)/(n_i + 128).$$

Note that $p_i(x)$ is a mixture of the empirical distribution $(n_i(x)/n_i)$ and a uniform distribution, assigning non-zero probabilities even to bytes that are not present in the feature vectors in training data. This provides a robust performance should such a byte appear later in a feature vector when the model is used for classification. Using our independence assumption in the model, the estimated conditional probability that the feature vector $\mathbf{x}$ occurs given that the flow belongs to class i is computed as

$$p_i(\mathbf{x}) = \prod_{j=1}^{nN} p_i(x_j). \tag{1}$$

MARKOV: Here we fit a first-order Markov-model to the features where the distribution of the first byte $p_{i,1}(x_1)$ as well as the transition probabilities $p_i(x_{j+1}|x_j)$ are estimated by the corresponding empirical frequencies in the training data for class i. The conditional probability of a vector $\mathbf{x}$ can be formulated as

$$p_i(x) = p_{i,1}(x_1) \prod_{j=1}^{nN-1} p_i(x_{j+1}|x_j). \tag{2}$$

The same estimator was used in [2].

MARKOVKT: In this method we also fit a first-order Markov-model, but we apply the Krichevsky-Trofimov estimate in all cases instead of empirical frequencies.

CONTEXT TREE WEIGHTING METHOD (CTW): CTW is a state-of-the-art lossless data compression algorithm [13] that fits a variable-order Markov model (more precisely, a mixture of such models) to the data which can be used to estimate the conditional probability $p_i(\mathbf{x})$ of a feature vector $\mathbf{x}$ for each traffic class i.

A variable-order Markov model can be described by a complete tree: The edges of the tree are labeled with source symbols, and the edge labels of a path to a node of the tree (from the root) determine a context (the context is the empty string at the root, and is of length d at nodes of depth d). Each leaf is labeled with its context and is also associated with a probability distribution that describes the distribution of the next source symbol immediately following the context specified by the leaf. That is, given the source data so far, one finds the leaf whose context is a suffix of the data, and the next symbol is distributed according to the distribution specified at the leaf. Using variable-memory Markov models can decrease complexity as well as improve prediction performance if some context do not occur sufficiently many times in the training data. The CTW method averages the predictions of all possible tree predictors (of a given maximum depth) in such a way that it guarantees that the performance of the averaged predictor is almost as good as that of the best tree predictor selected in hindsight,. The main trick is in the implementation: CTW achieves this goal by building a single full tree in a clever way, making the algorithm practically implementable. Nevertheless, large alphabet sizes (as the case of bytes) and deep trees may still require significant memory, and the memory footprint is the typical bottleneck in practical applications of the CTW method. In our experiments the maximum depth of the tree was set to 5, allowing at most five-character-long contexts or, in other words,at most fifth-order models.

RANDOM FOREST (RF) is one of the most widely used classification method [14] that averages the estimates of several decision trees. Each individual decision tree is built on a bootstrap sample of the original data, separating randomly selected leafs according to feature values that provide the best split among a given number of randomly selected features until each leaf contains only one feature point (in our case the features are the bytes of the feature vector $\mathbf{x}$). Using several decision trees at the same time can improve the classification accuracy significantly and may make the method more robust against noise. However, we have to note that RF is prone to overfitting, for example, if the training data set is chosen inappropriately.

B. Complexity of the Algorithms

Online traffic classification has great importance in practice from network management to enforcing service terms. Since the devices running these algorithms generally have very limited resources (CPU and memory) and the incoming high speed network traffic must be handled in real time, it is necessary to analyze the storage and computational complexities of the proposed techniques. According to their complexities, our methods can be split into three major categories: the KT-ESTIMATOR is based on a zero-order model, MARKOV and MARKOVKT are first-order ones, while RANDOM FOREST and CONTEXT TREE WEIGHTING belong to the class of higher-order models.

In case of a zero-order model like KT-ESTIMATOR, it is enough to store the probabilities or empirical counts for all the bytes occurred in the payloads. For each protocol, the algorithm needs to store at most 256 probability values only (one for each byte value). As we can see in the following sections, this very simple and lightweight solution shows surprisingly good precision in practice.

First order methods require significantly more, but still very little memory, as they need to store the distribution of the first byte of the feature vectors, and the transition probabilities of that a given byte is followed by another one in the features. This means that the model can be described using at most $256 + 256^2 = 65792$ probabilities. However, the byte streams generated by different network protocols follow specific rules and structures, resulting in very sparse transition matrices in practice, which can be utilized to reduce the memory requirements for storing a model.

Considering the CTW method, in the worst case one needs to store a full tree with branching factor 256 and depth D, requiring memory of order $\sum_{i=0}^{D} 256^i$ (we set $D = 5$ in our experiments). However, in practice, the size of the context trees are orders of magnitudes smaller than this theoretical upper bound, since only a small portion of byte sequences appear in the protocol specific data. Note that in all the above cases the (worst-case) memory complexity of our algorithms is independent of the length of the feature vectors. To reduce the high memory requirements of CTW, several extensions were considered including techniques to cut off paths that no longer split up, customize its operation for ASCII byte sources, store logarithms of ratios of conditional probabilities instead of actual block probabilities and use hashing to efficiently access the context tree data stored in memory.

The space complexity of RANDOM FORESTS is mainly affected by two parameters, namely, the number of attributes which is the number of input bytes (nN) and the number of trees to be evaluated. In our experiments, 10 trees have been used with $\log nN + 1$ attributes used to determine the decision at a node in a tree. The practical implementations of RF use further techniques to limit the maximal depth of the individual decision trees, yielding that the memory footprint of this solution can practically be kept on a constant level (independent of the length of the features).

The computational complexity of each examined algorithm is linear in the number of input bytes. This is obvious for our zero and first order models where Eq. 1 and Eq. 2 have to only be evaluated. However, for the CTW method, we first have to find the right probability value by searching in the context tree. Since the depth of the context tree is limited to 5, finding the probability takes $O(nN)$ steps only. Finally, the evaluation of a RF depends on two factors, the number of trees as well as their depths. Since we set both to some constant values, evaluating all the trees in a RF takes only $O(nN)$ time steps. The methods using maximum likelihood estimation (KT-ESTIMATOR, MARKOV, MARKOVKT, CTW) need to compute the likelihood for all the C protocol models before making a decision. In contrast, RF has the advantage that it requires only one model evaluation; however, the size of the trees are implicitly effected by the number of protocol models.

Note that both the computational and memory complexities of our algorithms are very little relative to the widely used automaton-based DPI solutions. These methods rely on signature databases that contain regular expressions describing protocol specific contents. These expressions are used to build a finite-state automaton that matches these regular expressions on the payload bytes. To construct these automata, several methods have been developed so far, using non-deterministic or deterministic finite automaton (NFA or DFA) or some hybrid solutions. For a large number of patterns, the evaluation of a NFA can be computationally very expensive because a large number of states has to be evaluated for each byte in parallel. On the other hand, a NFA can be transformed to a DFA having linear evaluation time, but, as a side-effect, it often causes an exponential explosion of the state-space, resulting in a 2^n-state DFA for a NFA with n states. In addition, there are methods [15] that provide a good trade-off between the memory used and the required computational time, but there are no methods that can optimize both at the same time. In contrast to standard DPI approaches, our techniques have limited memory requirements and their computational complexity is proportional to the number of examined traffic classes only and not to the number of used signatures which is usually much more.

III. DATA COLLECTION IN A FULLY CONTROLLED ENVIRONMENT: ACTIVE TRACE GENERATION

Collecting ground truth data is a crucial part of traffic classification studies. A generally prevailing method in the literature is to collect (unlabeled) traffic traces from different sources (network operators, campus networks, testbeds, etc.), and then to label each flow with the generating protocol or application based on some DPI tools or other heuristics. The obvious advantage of this method is that the resulting labeled traces are real since DPI tools hardly result false positive outcomes. On the other hand, the result of the labeling phase may be incomplete due to the shortcomings of DPI engines, and significant portion of the applications/protocols may be dropped out from the analysis because of their small incidence or because the applied labeling method is not able to recognize them (see, e.g., [16]). The quality of the data set generated in this way largely depends on the quality of the applied DPI tool itself, which varies in a wide range from freely available solutions with low precision to very expensive and accurate commercial tools.

To alleviate the shortcomings of DPI-based labeling for ground truth, we used an active data collection technique: traffic traces were captured in a fully controlled environment where a modified kernel module logged each flow and the coresponding appliacation; this methodology results in an

Application Class	WIRELESS		LAN	
	Flows	Bytes	Flows	Bytes
PPLive	0	0 MB	5100	2100 MB
PPStream	6900	538 MB	5700	3545 MB
SopCast	1700	608 MB	48000	24855 MB
TVUPlayer	3250	1365 MB	2800	1872 MB
BitLord	1900	2253 MB	26500	23923 MB
Emule	8700	540 MB	59500	6662 MB
LimeWire	2100	2585 MB	2500	1443 MB
others	0	0 MB	30000	42000 MB

TABLE I
THE AMOUNT OF TRAFFIC IN WIRELESS AND LAN.

unquestionable source for the ground truth (similar active data collection can be found, e.g., in [11], [17]).

In this work we concentrate on P2P protocols, as they are less studied than traditional protocols (like HTTP, FTP, etc.), play an increasing role in the Internet, and have become responsible for a large portion of Internet traffic recently. We have generated and recorded traffic of different P2P applications in two main groups: P2PTV clients and P2P file-sharing clients. The P2PTV group contains four clients with different proprietary protocols (PPLive, PPStream, SopCast, TVUPlayer), while the group of file-sharing applications consists of three client applications with different protocols (BitLord as a BitTorrent client, Emule as an eDonkey client and LimeWire as a Gnutella client). To be able to analyze the performance of our algorithms in the presence of significantly different traffic types, traces of other applications, such as HTTP, Skype, gaming, encrypted torrent, etc., have also been collected; these traces are used to create a general class, called *others*, to test how much our algorithms can distinguish the learnt traffic types from unseen ones.

Two data sets are used during the evaluation: a smaller set, WIRELESS, consisting of traces with full payload, recorded in wireless environments (WiFi and 3G), and a larger set, LAN, recorded over high-speed LAN connection during another experiment, containing only the first 16 bytes of the payload of each packet. A detailed description of the recorded traffic, showing the number of flows and the amount of traffic carried, is given in Table I.

In the next section we demonstrate the feasibility of our early traffic classification approach on the combined data set composed of WIRELESS and LAN (without the *others* class), while refinements of our methods, concerning the number of packets and the amount of payload to be considered will be tested only on WIRELESS. We also use the combined data set (including the *others* class) to test how the algorithms deal with unknown traffic types and asymmetric routing (when only one direction of the traffic flow is observable), while robustness to changing network environments is examined using the WIRELESS data as training and the LAN data as test set.

IV. LABORATORY EXPERIMENTS

In this section we present experimental results on the WIRELESS and LAN datasets. We used two main metrics: the true positive (TP) and the false positive (FP) ratio. The TP ratio is the proportion of successfully classified samples to all samples, while the FP ratio is the proportion of samples falsely classified to a given class to all samples that do not belong to that class. Both metrics are usually computed for both flows and bytes. Each experiment was repeated ten times on randomly selected training and test data (of a given size).

A. A feasibility test: classification from the first 16 bytes of the first packet of each flow

To demonstrate the feasibility of our approach and improvements over the memoryless and first-order Markovian methods of [2], we tested the proposed methods on the large, combined data set of WIRELESS and LAN, using the first 16 bytes of the first packet of each flow. The TP and FP ratios (using ten-fold cross-validation) are shown in Figure 1. One can observe that higher order models usually perform better. The only exception is the simple first-order MARKOV model (also used in [2]) whose performance is degraded by the inadequate handling of patterns not observed in the training data, which is corrected by using the Krichevsky-Trofimov estimator in MARKOVKT. The RF method achieved a remarkable $98-99\%$ average TP ratio (in bytes), and the CTW and MARKOVKT methods also achieved very good average TP ratios of about $95-96\%$. The performance is also attractive if we consider the FP ratio, as the higher order methods achieved FP ratios far below 5%, with the exception of CTW in case of PPStream with 6%. The classification results on LimeWire worth some considerations: as the UDP version of this application starts with a 16-byte random string, its identification is clearly impossible from our data, while the TCP versions use some hand-shake mechanism, which is identifiable. The error rates in identifying this application are proportional to the share of UDP traces in our data.

We also compared our algorithms to different publicly available DPI methods. Since they are developed to identify traffic types from full payload, we performed these tests on the WIRELESS data set. We believe that this comparison also provides some important insights, although it is not fair from either side: (i) the DPI algorithms are designed to solve the harder problem of recognizing more traffic types, and it is likely that the performance of our algorithms would degrade if the number of traffic types were increased; (ii) the DPI algorithms can use more information than ours; (iii) freely available DPI tools are less accurate than the commercial ones widely used in practice.

We used two popular DPI tools, OpenDPI [18] and Tstat [19], as well as the payload-based classifier of [10], called Coral in this paper, which served as the source of ground truth in the large-scale experiments of [4]. Coral applies host level identification if the payload inspection fails; to make a fairer comparison, we also tested the tool without this component, solely relying on payload data (Coral*). The results are given in Figure 2 for both flow and byte accuracy. For simplicity, we only provide comparison to our best algorithm, RF (based on the first 16 bytes of the first packet of each flow), whose ten-fold cross-validation performance is also reported (the results for CTW and MARKOVKT are similar). Moreover, in case of the DPI algorithms we only provide results for the traffic

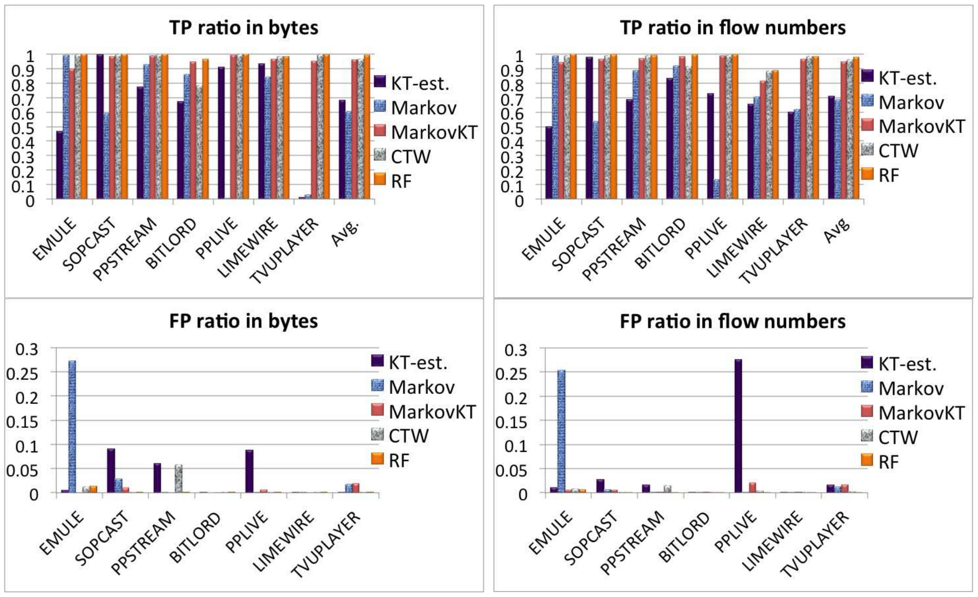


Fig. 1. Comparison of the different machine learning methods on the aggregated data set.

classes they support, and report the class *others* whenever the DPIs classify to a traffic class that does not belong to our data set (this explains the increased false positive values for the *others* class).

The results show that the performance of RF is usually at least as good as those of the DPI methods in TP ratio, while the DPIs are usually better in FP ratio. The exceptions to this finding are the results of Coral and Tstat for BitLord. The errors made by Coral are due to the usage of its host based early decisions; this approach helps with real traffic traces but is misled by the artificial patterns in our measurements. Turning off this feature (i.e., Coral*) yields improved FP results. However, we have no good explanation of the high FP ration for BitLord in case of Tstat. As a side effect of our (admittedly limited) measurements one can see that a careful test of DPI tools is needed on a given data set if they are to be used as the source of ground truth for real traffic traces.

B. Bytes and packets: how much data is needed?

In this section we examine how the accuracy of our classifiers depends on the amount of data used. We test how the results change with the number of packets and bytes used from each flow, as well as how fast the algorithms learn, that is, how many training flows are needed to achieve good performance. Since in these experiments we often use payload beyond the first 16 bytes of each packet, we have run the tests on the WIRELESS data set. Due to space limitations, results are only provided for our best algorithm, RF (our other methods show similar characteristics).

Figure 3 shows the performance of the RF classifier (using ten-fold cross-validation) as a function of the number of packets n and number of bytes N used from each packet (in this experiment symmetric routing is considered, that is, the classifier is able to observe both directions of the flows and the packets are selected sequentially from both directions). Experiments have been performed for $n = 1, \ldots, 6$ and $N = 1, \ldots, 72$, the latter being justified by the assumption that protocol specific headers are usually contained at the beginning of each packet. One can see that increasing the number of bytes N sharply improves the performance at the beginning, which flattens out around 8 bytes, while increasing the number of packets may actually worsen the performance in the latter region, which may be attributed to the increased feature-dimension of the problem that eventually leads to overfitting.

One can also see from the above experiment that using the first packet of each flow is sufficient (this choice also has the advantage that it seems pretty much independent of the network architecture as the content of the first packet usually does not depend on the underlying network infrastructure). To be able to estimate the necessary amount of data to be collected from different traffic classes, we tested how many training flows are needed to achieve good performance (as a function of the number of bytes used from the first packet). The classification accuracy of the RF algorithm is given in Figure 4 for different training set sizes: 50, 100, 200, and 400 flows from each class are used as training data, respectively, and the remaining flows are used for testing. Each experiment was repeated ten times, using a random selection of the training data. The results of the tests with ten-fold cross-validation are also reported.

One can see that using only a few hundred training flows results TP rates above 90%. Also, for all training set sizes the performance improves in a similar manner as observed in Figure 3, increasing between 1 and 8 bytes and then smoothing out. We can also notice that the byte-based measures sharply improve around 3 bytes. The reason for this observation may be that only a small portion of protocol messages is responsible for real data transfer (i.e., it is easier to identify flows carrying large amount of data), or it may just simply be an artifact of our data set (where certain patterns randomly occur just for a

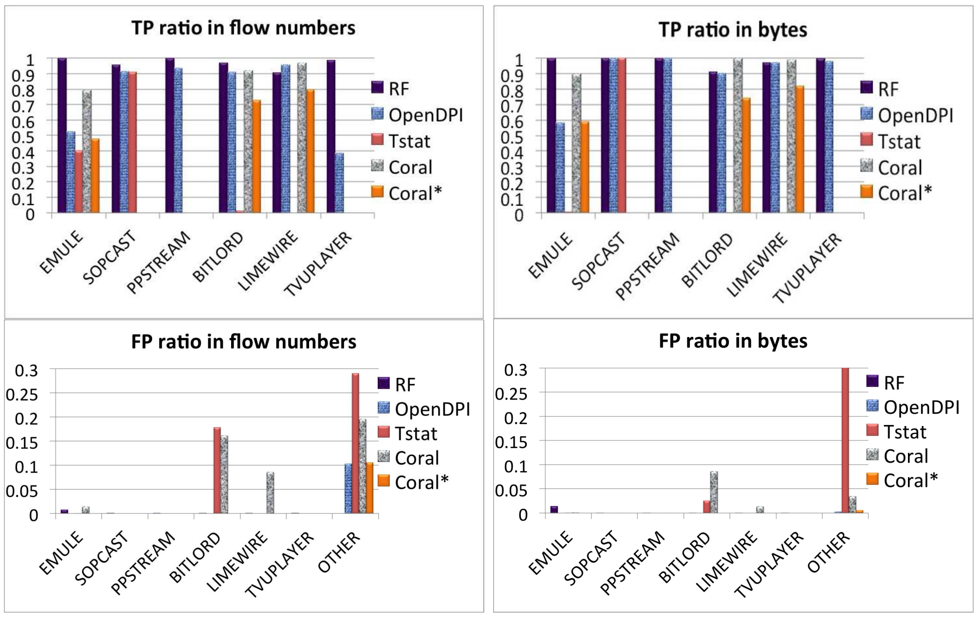


Fig. 2. The performance of the RANDOM FOREST classifier using the first 16 bytes of the first packet of each flow, compared to some DPI tools.

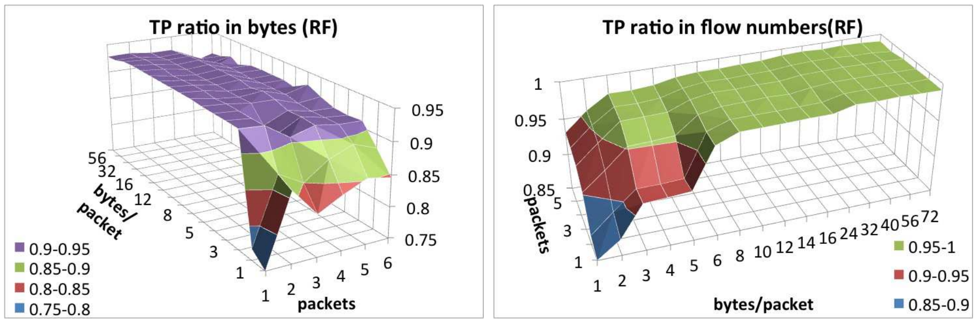


Fig. 3. The TP ratio of the RANDOM FOREST classifier as a function of the number of packets n and bytes N used ($N = 1, \ldots, 72$, $n = 1, \ldots, 6$).

few elephant flows).

Figure 5 presents the TP and FP ratios (in flow number) for the different protocol classes in the ten-fold cross-validation measurement. We can observe that the necessary amount of information depends on the protocol class, for example, the Gnutella-client LimeWire requires more data than other protocols (this is in agreement with earlier findings of [10], although it seems that using around 30 bytes of the payload is sufficient in contrast to the 300-400 bytes observed in [10], but this may also be the result of not having enough LimeWire flows in our data sets).

C. Robustness

In a real network environment it is expected that the classifier has to deal with a lot of traffic that need not be classified, or several protocols and applications unknown to the classifier. Treating such situations inappropriately leads to a large increase in the FP rate. To reduce this effect, we introduced a new class, called *others*, composed of traces from diverse applications (Skype, encrypted BitTorrent, HTTP, etc.) that represents unknown traffic types. Then the problem is extended to classify a flow to one of the earlier mentioned P2P protocols, or to *others*.

The effect of this modification was measured (using ten-fold cross-validation) on the combined data set composed of WIRELESS and LAN, including the *others* traffic class, using the first 16 bytes of the first packet of each flow. A comparison of the results, presented in Figure 6, to the case without the *others* class (cf. Figure 1) shows that the TP ratio (for the better algorithms) is only slightly decreased for most of the P2P protocols, with the exception of Emule that is often confused with traffic from the *others* class. This is also reflected in the FP ratio, where high values can mostly be observed for Emule (and for PPLive in case of the MARKOV classifier, as before). The classification results are the worst for the *others* class in both TP and FP ratios (worsening substantially the average TP ratio), but, in fact, here we are most sensitive to the errors that classify *other* traffic to a known P2P traffic (which is reflected in the FP rate of the P2P traffic classes). The latter type of error seems to be substantial for the RF algorithm, which has the best performance in the tests without the *others* class. The reason for this performance degradation is that the *others* class is quite diverse, and it happens that our classification algorithms see a certain traffic type only in the

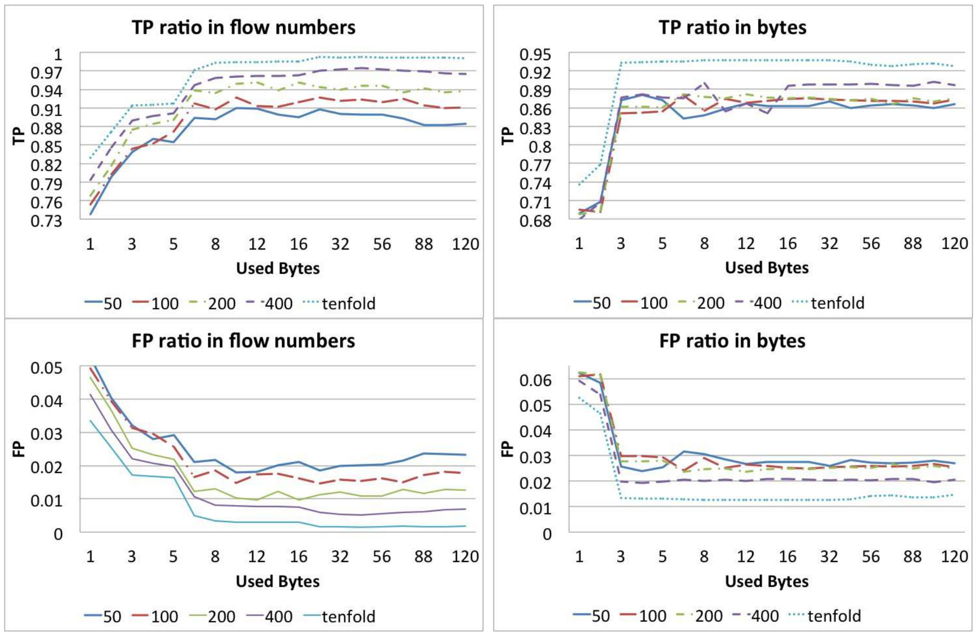


Fig. 4. The average TP and FP ratios as functions of the number of bytes used from the first packet for different training set sizes in case of the Random Forest classifier.

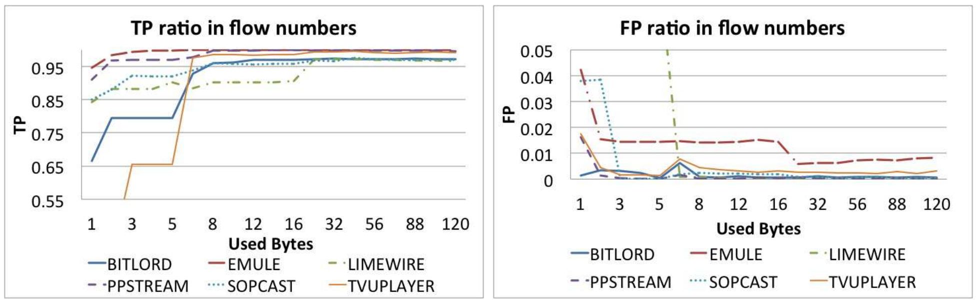


Fig. 5. The TP and FP ratios for the different protocol classes (for flows) as a function of the number of used bytes in case of Random Forest.

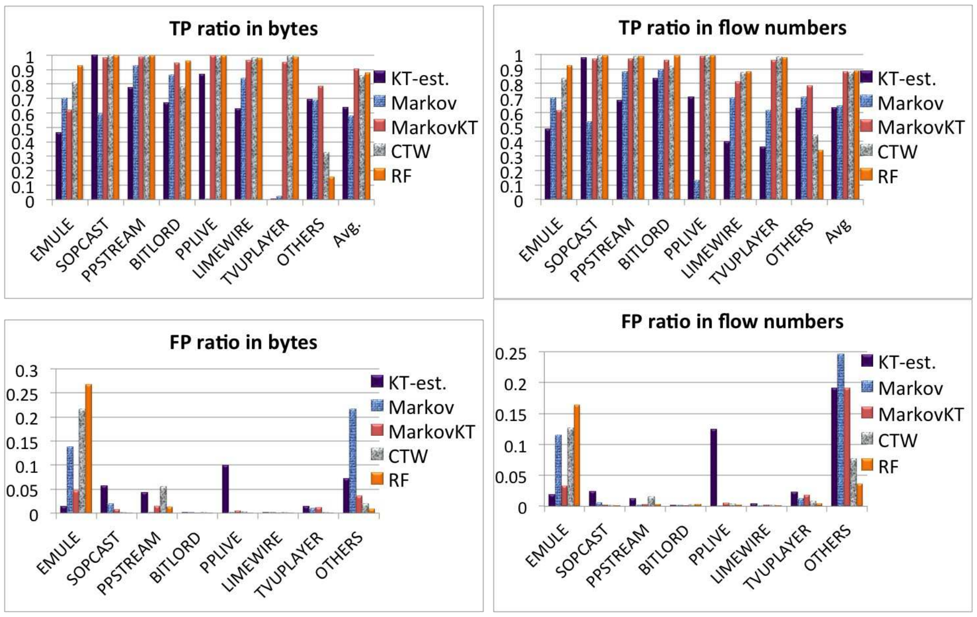


Fig. 6. Comparison of the different machine learning methods on the aggregated data set extended with the *others* class.

test set, in which case such traffic may easily get misclassified.

Another important issue in traffic classification, which has to be taken into account, is the asymmetric nature of routing, for example, in backbone networks. This phenomenon often yields that we are able to observe traffic from only one direction of a flow. To handle this situation, we can apply our method to the first few payload bytes of the first backward packet as well (our previous single-packet results always correspond to the forward direction). Ten-fold cross-validation performance of the resulting 16-byte RF classifier on the combined WIRELESS-LAN dataset is shown in Figure 7. The results indicate that the method is quite applicable for this scenario and results only a $1-2\%$ performance drop compared to the classification results based on the forward packets.

Traffic classification algorithms often suffer from the problem that they behave differently under different network conditions and use features that are specific to the underlying networks (e.g., connection type). We expect that this is not the case with our algorithms, as the first 16 bytes of each flow are unlikely to carry network specific information. To see how our methods work in changing network environments, we trained an RF classifier on the WIRELESS data set, and tested it on LAN. The two data sets were recorded at different times with different computers and connection types, so our classifier can only utilize protocol specific similarities. The tests show very similar performance to the one obtained for the combined data set, with an approximately 10% drop in the TP rate for LimeWire (corresponding to the changed proportion of UDP traffic), while the results for the other classes remained essentially the same. In case of LimeWire, using some more bytes may significantly improve the result, as indicated in Figure 5.

D. The scale and level of noise

In this section we present experimental results concerning how our algorithms work when the training data is not fully reliable and is affected by errors. To simulate the presence of noise, flows with incorrect labels generated by different applications were added to the training sets. The performance degradation as a function of noise level using the MARKOVKT method is shown in Figure 8 (using the first 16 bytes and ten-fold cross-validation on the WIRELESS dataset); other classifiers examined in the paper show very similar results.

One can observe that the presence of 10% noise in the training data slightly decreases the avergage accuracy of MARKOVKT: for TP ratio the degradation is around 1% while it is even less for in case of false positives. After a 10% noise level the performance drops much faster, but our methods achieve a surprisingly high overall accuracy even if almost half of the training data is labeled incorrectly: at 40% noise level the true positive ratio remains still above 84% in flow numbers and above 88% in byte counts, meanwhile the false positive values deteriorate even more slowly, being less than 3% and 2% in flow numbers and byte counts, respectively. The reason behind this phenomenon is that our algorithms are able to filter out the statistically irrelevant information added by noises in the training set, and even in the presence of such an erroneous data the proposed models perform very well. The main advantage of the techniques based on statistical analysis of the payload is that they are generally less prone to tiny inconsistencies and noises in the input data.

Application Class	Flows	Bytes
BITTORRENT	209764	8447 MB
DC	232	1797 MB
DNS	19819	8 MB
FTP	80	265 MB
GNUTELLA	29776	28 MB
HTTP	89961	4145 MB
IMAP	219	9 MB
POP3	875	29 MB
PPSTREAM	628	139 MB
RTMP	56	44 MB
RTP	139	51 MB
RTSP	90	429 MB
SIP	454	1 MB
SMTP	95	14 MB
Souce engine	2743	0.7 MB
SSH	37	0.1 MB
WAP	865	19 MB
WINDOWS	560	0.1 MB
WOW	17	12 MB
XBOX	12	20 MB
XMPP	66	0.4 MB
Total	356.8 k	15.5 GB

TABLE II
THE AMOUNT OF TRAFFIC IN THE REAL WORLD DATA SET.

V. VERIFICATION ON REAL WORLD TRAFFIC TRACES

After carrying out laboratory experiments, we have also verified our models on real world traffic traces collected in the network of a large European ISP. To label the traffic with real application classes a commercial DPI tool developed by one of our industrial partners is applied, which is far more reliable than the publicly available ones. Since this real data set contains more realistic and diverse network flows generated by a dozen different applications, it enables a more complex validation of the proposed techniques. The various protocols and their proportion in the whole set can be seen in Table II. One can observe that although the data consists of hundred thousands of flows carrying approximately 16 GB data, in contrast to our laboratory traces, it is very unbalanced. There are protocol classes like SIP and XMPP that are underrepresented while others, such as HTTP and BitTorrent, have high presence in the overall traffic.

This unbalanced property makes the separation of data into train and test sets difficult. For overrepresented protocol classes, a few hundreds flows have been considered as training data, while in the other cases the limited number of data has been split into two sets of equal size. The relationship between the number of bytes used and the true and false positive ratios for the various methods is shown in Figure 9. Similarly to the laboratory experiments, one can observe that, in spite of its simplicity, the very lightweight MARKOVKT method shows promising classification accuracy, in both true and false positive ratios. Furthermore, we can also see that using only the first 32 bytes of the first packet in each flow is enough to achieve an outstanding performance.

Next, we focus on MARKOVKT and use only the first 32 bytes of the network flows. The confusion matrix for the

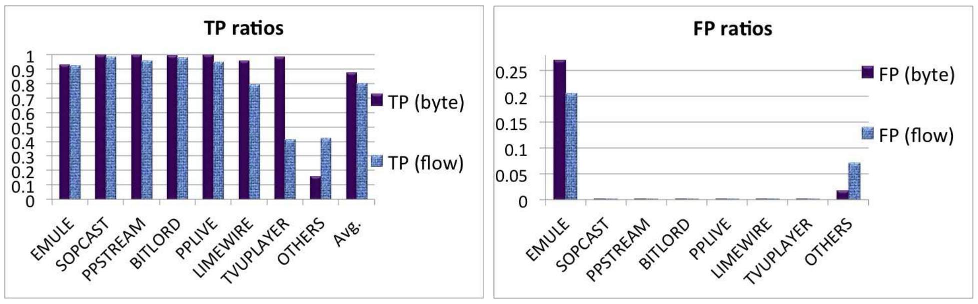


Fig. 7. The performance of the RANDOM FOREST classifier using the first 16 bytes of the first backward packet of each flow.

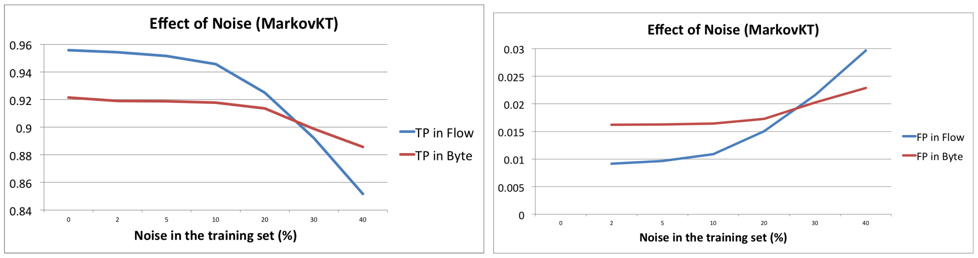


Fig. 8. The performance of the MARKOVKT classifier if the training data is unreliable or noisy.

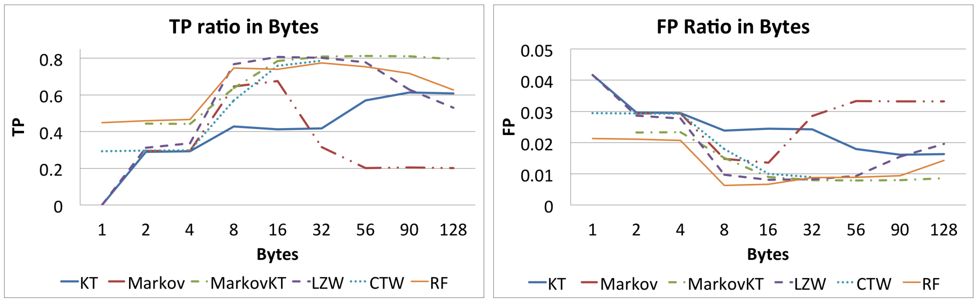
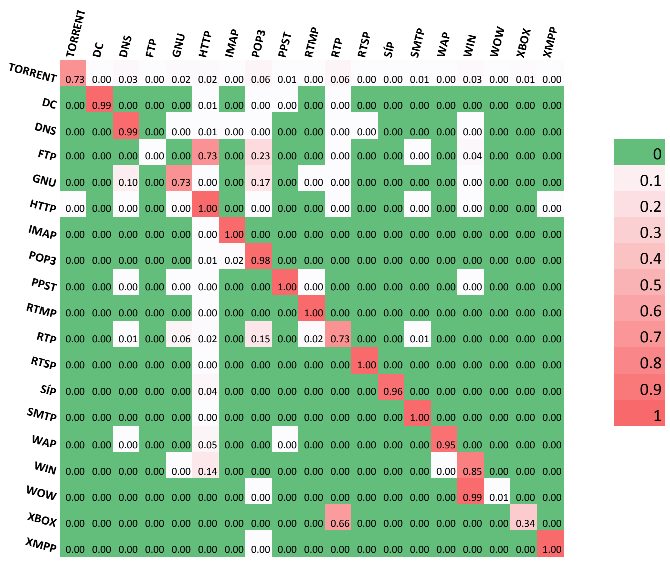


Fig. 9. True and False Positive ratios for feature vectors with various lengths on real traffic traces.

aggregated results is shown in Figure 10 where the vertical axis is labeled with the real traffic classes while the horizontal axis shows the estimates of MARKOVKT. One can observe that almost all the large values are in the main diagonal, indicating high classification accuracy. Only a few relevant outliers belonging to FTP, XBOX and WoW traffic can be identified. To find out the real reason behind this phenomena, we have analyzed these classes separately.

For each of these protocols, the number of flows in the training set is very limited, for example in case of FTP, XBOX and WoW only 30, 5 and 7 flows are considered, respectively. Figure 11 shows the true positive ratios together with the training set sizes. There is a direct correlation between the number of training flows and the corresponding true positive ratio. The picture is quite similar for false positive values. Considering FTP traffic, another issue can be raised, namely, the majority of the traffic belong to data streams and not to control ones, for which the payload does not carry any protocol specific markers, but the user content only. This real world experiment shows that the proposed very lightweight approaches could successfully be applied even in presence of a large number of various protocols.



	TORRENT	DC	DNS	FTP	GNU	HTTP	IMAP	POP3	PPST	RTMP	RTP	RTSP	SÍP	SMTP	WAP	WIN	WOW	XBOX	XMPP
TORRENT	0.73	0.00	0.03	0.00	0.02	0.02	0.00	0.06	0.01	0.00	0.06	0.00	0.00	0.01	0.00	0.03	0.00	0.01	0.00
DC	0.00	0.99	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
DNS	0.00	0.00	0.99	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
FTP	0.00	0.00	0.00	0.00	0.00	0.73	0.00	0.23	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.04	0.00	0.00	0.00
GNU	0.00	0.00	0.10	0.00	0.73	0.00	0.00	0.17	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
HTTP	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
IMAP	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
POP3	0.00	0.00	0.00	0.00	0.00	0.01	0.02	0.98	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
PPST	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
RTMP	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
RTP	0.00	0.00	0.01	0.00	0.06	0.02	0.00	0.15	0.00	0.02	0.73	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00
RTSP	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
SÍP	0.00	0.00	0.00	0.00	0.00	0.04	0.00	0.00	0.00	0.00	0.00	0.00	0.96	0.00	0.00	0.00	0.00	0.00	0.00
SMTP	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00
WAP	0.00	0.00	0.00	0.00	0.00	0.05	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.95	0.00	0.00	0.00	0.00
WIN	0.00	0.00	0.00	0.00	0.00	0.14	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.85	0.00	0.00	0.00
WOW	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.99	0.01	0.00	0.00
XBOX	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.66	0.00	0.00	0.00	0.00	0.00	0.00	0.34	0.00
XMPP	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00

Fig. 10. The confusion matrix showing the proportion of correctly classified bytes.

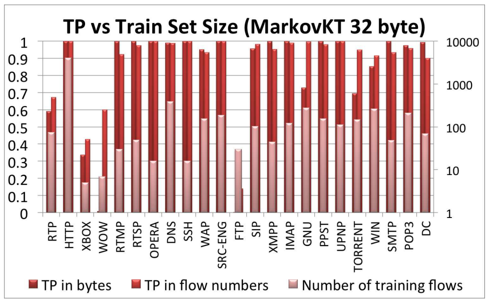


Fig. 11. The achieved TP ratios on the real network data compared with the training set sizes. (Note: training set sizes are displayed on logarithmic scale.)

VI. Limitations

In this section we discuss the limitations of our early-classification approach, which need to be addressed via different techniques. First of all, since our methods purely depend on the payload of the network traffic, encrypted network flows cannot be handled at all. On the other hand, the vast majority of today's Internet traffic is still unencrypted. Although the proportion of encrypted traffic will likely increase in the near future, it is expected that a significant portion of the total traffic will remain unencrypted, due to several reasons, for example, since encryption requires a lot of computational resources (hence quickly reduces the battery life of mobile devices) and is not necessary for most of the applications.

Another issue we have to deal with is related to encapsulation when the content of a given application is embedded into another trusted protocol like HTTP. Several applications use this technique to go through firewalls. If a protocol is embedded into simple HTTP, despite the unencrypted data the proposed techniques would identify the carrying HTTP instead of the real applications. Our methodology in its current form is not prepared for such situations when the payload does not start with the protocol specific content. However, this can be solved by the segmentation of HTTP/HTML messages. To this end, the embedded content has to be searched for, extracted, and than the proposed technique can easily be applied for the separate segments.

Furthermore, by their nature, our methods are vulnerable to countermeasures when a malicious user injects some arbitrary bytes into the beginning of the payloads to mislead the classifier. Fortunately, it is not the general case, and most of the users use the network protocols as they are. If, however, a malicious user modifies the payload, the accuracy of our methods may substantially degrade. This problem may be alleviated by looking for the most protocol specific parts of the payload during the training instead of using the first bytes of the flows. As we mentioned before, this issue is already present in case of UDP-based LimeWire traffic, where the first 16 bytes do not follow a well defined protocol specific pattern. This phenomena may have various reasons: the first bytes carry increasing sequence numbers, random identifiers, filenames, etc.

However, if the training set is large enough and contains accurate class labels, only a slight effect on the accuracy of our approaches is expected to have. Finally, we also have to mention that in commercial products it is not uncommon to have hundreds of application classes to be handled. Although the most widely used protocols are considered in the experiments of this paper, the protocol portfolio will certainly change in the future, and new protocols should also be considered. We expect that our methods would scale up well to such problems and achieve similarly good performance in general. Nevertheless, with the number of traffic classes increasing, our methods might become more sensitive to the above mentioned limitations (e.g., if more and more new protocols start with random strings).

In spite of the revealed limitations, the proposed techniques show sufficiently good performance even in real world environments. Furthermore, since they have extremely low computational and storage complexities and can easily be implemented even in hardware, they could serve as lightweight pre-classifiers that can efficiently label or even filter out wanted or unwanted traffic in the very early stage of a byte flow. If the methods are applied for prescreening, one can also consider a refined version of the algorithms where the classifiers also report the uncertainty of their label predictions (e.g., based on similar measurements as reported in Figure 10), and further, more resource-intensive methods can be applied to deal with the hard cases where the uncertainty of our algorithms is high.

VII. Conclusion and Future Work

In this paper we showed that effective classification of P2P traffic can be performed based on the first few bytes of the first packet of each flow. The proposed classifiers are based on standard stochastic models and state-of-the-art machine learning methods (such as random forests, Markov-models or context trees) and can reach a remarkable accuracy over 95% using as limited data as the first 16 bytes of the first (or the first backward) packet of each flow. This result is competitive to other state-of-the-art algorithms. The advantage of our method is that, unlike traditional DPIs, no human expertise is needed to design the appropriate signatures, and only a collection of sample flows from each class is necessary. The data collection step is especially easy since the sample flows may be generated in a completely artificial environment, alleviating the problem of obtaining the ground truth for traffic measurements made in real networks.

A thorough performance analysis was made on both laboratory and real world traffic traces to examine the accuracy of our algorithms. The laboratory tests were carried out on actively collected data: in this case the ground truth is known without doubt, but the traffic patterns are less realistic than in real traces. However, we believe that the fact that our algorithms use only the first 16 bytes of the flows makes our method less sensitive to such errors (apart from the fact that the TP and FP rates are highly dependent on the composition of the whole data set).

While during laboratory experiments only a few P2P protocols were taken into account, we also showed how the proposed techniques scale up to handle a large number of various protocol types that may appear in real world traffic. To this end, the proposed methods were also validated on a much more diverse data set collected in the network of a large European ISP, and achieved surprisingly good accuracy for the majority of the protocol types on these real traces. We also considered the problem of handling previously unseen traffic types, and showed training methods to deal with such problems.

Acknowledgments

We would like to thank H. Kim, D. Barman and T. Karagiannis for providing the code of the Coral classifier [4], [10] and G. Szabó and A. Finamore for the helpful discussions.

References

[1] A. Finamore, M. Mellia, M. Meo, and D. Rossi, "Kiss: Stochastic packet inspection," in *TMA '09: Proceedings of the First International Workshop on Traffic Monitoring and Analysis*, (Berlin, Heidelberg), pp. 117–125, Springer-Verlag, 2009.

[2] J. Ma, K. Levchenko, C. Kreibich, S. Savage, and G. M. Voelker, "Unexpected means of protocol inference," in *IMC '06: Proc. of the 6th ACM SIGCOMM conf. on Internet measurement*, (New York, NY, USA), pp. 313–326, ACM, 2006.

[3] T. T. T. Nguyen and G. Armitage, "A survey of techniques for internet traffic classification using machine learning," *IEEE Communications Surveys and Tutorials*, vol. 10, no. 4, pp. 56–76, 2008.

[4] H. Kim, K. Claffy, M. Fomenkov, D. Barman, M. Faloutsos, and K. Lee, "Internet traffic classification demystified: myths, caveats, and the best practices," in *CoNEXT '08: Proceedings of the 2008 ACM CoNEXT Conference*, (New York, NY, USA), pp. 1–12, ACM, 2008.

[5] M. Pietrzyk, J.-L. Costeux, G. Urvoy-Keller, and T. En-Najjary, "Challenging statistical classification for operational usage: the adsl case," in *IMC '09: Proc. of the 9th ACM SIGCOMM conf. on Internet measurement*, (New York, NY, USA), pp. 122–135, ACM, 2009.

[6] T. Karagiannis, K. Papagiannaki, and M. Faloutsos, "Blinc: multilevel traffic classification in the dark," in *SIGCOMM '05: Proc. of the 2005 conf. on Applications, technologies, architectures, and protocols for computer communications*, (New York, NY, USA), pp. 229–240, ACM, 2005.

[7] M. Iliofotou, H.-c. Kim, M. Faloutsos, M. Mitzenmacher, P. Pappu, and G. Varghese, "Graption: A graph-based p2p traffic classification framework for the internet backbone," *Comput. Netw.*, vol. 55, pp. 1909–1920, June 2011.

[8] B. Gallagher, M. Iliofotou, T. Eliassi-Rad, and M. Faloutsos, "Link homophily in the application layer and its usage in traffic classification," in *INFOCOM'10: Proceedings of the 29th Conference on Information Communications*, (Piscataway, NJ, USA), pp. 221–225, 2010.

[9] L. Bernaille, R. Teixeira, and K. Salamatian, "Early application identification," in *CoNEXT '06: Proceedings of the 2006 ACM CoNEXT conference*, (New York, NY, USA), pp. 1–12, ACM, 2006.

[10] T. Karagiannis, A. Broido, M. Faloutsos, and K. Claffy, "Transport layer identification of P2P traffic," in *IMC '04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, October 2004.

[11] D. Rossi and S. Valenti, "Fine-grained traffic classification with netflow data," in *IWCMC '10: Proceedings of the 6th International Wireless Communications and Mobile Computing Conference*, (New York, NY, USA), pp. 479–483, ACM, 2010.

[12] R. E. Krichevsky and V. K. Trofimov, "The performance of universal encoding," *IEEE Trans. Inform. Theory*, pp. 199–207, Mar. 1981.

[13] F. M. J. Willems, Y. N. Shtarkov, and T. J. Tjalkens, "The context-tree weighting method: Basic properties," *IEEE Trans. Inform. Theory*, vol. IT-41, pp. 653–664, May 1995.

[14] L. Breiman, "Random forests," *Machine Learning*, no. 1, pp. 5–32, 2001.

[15] Y.-H. Yang and V. K. Prasanna, "Space-time tradeoff in regular expression matching with semi-deterministic finite automata," in *INFOCOM, 2011 Proceedings IEEE*, pp. 1853–1861, IEEE, 2011.

[16] G. Szabó, D. Orincsay, S. Malomsoky, and I. Szabó, "On the validation of traffic classification algorithms," in *Passive and Active Network Measurement* (M. Claypool and S. Uhlig, eds.), vol. 4979 of *Lecture Notes in Computer Science*, pp. 72–81, Springer Berlin / Heidelberg, 2008. 10.1007/978-3-540-79232-1_8.

[17] "Napa-wine project webpage." http://www.napa-wine.eu.

[18] "Opendpi webpage." http://www.opendpi.org/.

[19] A. Finamore, M. Mellia, M. Meo, M. M. Munafò, and D. Rossi, "Live traffic monitoring with tstat: Capabilities and experiences," in *Proc. 8th International Conference on Wired/Wireless Internet Communications (WWIC 2010)*, (Luleå, Sweden), pp. 290–301, 2010.