An Experimental Evaluation of the Computational Cost of a DPI Traffic Classifier

An Experimental Evaluation of the Computational Cost of a DPI Traffic Classifier

Niccolò Cascarano*, Alice Este†, Francesco Gringoli†, Fulvio Risso*, Luca Salgarelli†
* Dipartimento di Automatica e Informatica
Politecnico di Torino, Italy
Email: <name.surname>@polito.it
† Dipartimento di Elettronica per l'Automazione
Università degli Studi di Brescia, Italy
Email: <name.surname>@ing.unibs.it

***Abstract*—A common belief in the scientific community is that traffic classifiers based on Deep Packet Inspection (DPI) are far more expensive in terms of computational complexity compared to statistical classifiers. In this paper we counter this notion by defining accurate models for a Deep Packet Inspection classifier and a statistical one based on Support Vector Machines, and by evaluating their actual processing costs through experimental analysis. The results suggest that, contrary to the common belief, a DPI classifier and an SVM-based one can have comparable computational costs. Although much work is left to prove that our results apply in more general cases, this preliminary analysis is a first indication of how DPI classifiers might not be as computationally complex, compared to other approaches, as we previously thought.**

I. Introduction

Traffic classification has been one of the hottest research topics in recent years. With the decline in effectiveness of classifiers based on the examination of transport-layer ports, Deep Packet Inspection (DPI) techniques have emerged. Although these techniques are usually extremely precise (provided that traffic is not tunneled or encrypted) and are able to recognize a large number of different protocols, their biggest problem is perhaps the common belief that payload-based methods are extremely expensive in terms of CPU cycles and memory requirements, while the recently developed statistical techniques are thought to be less demanding [3]–[8]. However, despite run-time performances are an essential aspect for evaluating the capability of a classifier to process traffic in real-time even in presence of high-speed links, no serious investigations have been performed so far in this respect.

This paper aims at filling this gap by analyzing the computational complexity of two traffic classifiers, a software implementation of a DPI classifier and a statistical classifier based on Support Vector Machines (SVM) [15]. We carry out the analysis by modeling the two classification algorithms under examination in functional blocks. Since a mathematical comparison of the two models is not possible because they are based on different parameters, we decided to evaluate the computational cost of each block of our models by dissecting the code related to their implementation. Finally, we derive the overall costs by measuring the frequency of execution of each block by running each classifier on traffic traces captured on real networks.

This work was supported in part by a grant from the Italian Ministry for University and Research (MIUR), under the PRIN project *RECIPE*.

The results we present here are preliminary for two reasons. On one hand, we do not analyze the cost in terms of memory usage because this paper focuses on software-based implementations, where usually memory consumption is not an issue. On the other hand, only one category of statistical classifiers is analyzed (SVM-based), whereas a more thorough evaluation would need to consider other, possibly less computationally intensive ones (e.g., Naive Bayes).

Despite the above limitations, the results presented in this paper are significant, because they show that in real-world, usable implementations, DPIs and statistical classifiers can indeed lead to comparable computational costs, which goes against the belief most researchers in this area, ourselves included, have held so far.

This paper is organized as follows. Section II is dedicated to the related work, while Section III gives a brief introduction about the selected classification methods. Section IV presents the methodology used in the evaluation of the two classifiers, while Section V illustrates the most important characteristics of our implementation of the classifiers. Finally, Section VI presents and analyzes the experimental results, while Section VII concludes the paper.

II. Related work

The experimental evaluation of the complexity of traffic classification algorithms is a relatively unexplored topic. A first work [10] evaluates five machine learning algorithms, based on Naive Bayes, C4.5, Bayesian Network and Naive Bayes Tree. The comparison is performed on different clustering algorithms but using the same information and the same features extracted from the traffic. They measured the computation time of the WEKA implementation [12] of these algorithms, normalized to the fastest one; both the time of building protocol models and classification speed are analyzed.

A recent work [11] evaluates the performance of different statistical classifiers: host-behavior based [13] and various ses-

sion features based systems, using also in this case the WEKA implementation for the latter. They show the performance of the classifiers as function of the number of the training set samples, both in terms of accuracy and classification time. Both [11] and [13] use DPI to establish the ground truth of the protocols for the traces used in their experimental analysis, but its complexity is not evaluated.

Many other works have presented and analyzed statistical classification mechanisms in the recent past, including [3]–[8]. In all cases, the assumption behind these works is that DPI is too computationally complex to be used in real time systems; hence it is utilized mostly for post-processing traffic traces (off-line) to derive ground truth information.

III. Traffic classifiers under examination

This Section introduces the two classifiers under examination. The DPI classifier is based on regular expressions and strictly follows our own previous work [2]. Fhe SVM statistical classifier [9], [14] is based on a single class SVM algoritm [16], which was chosen for several reasons. First of all, SVM traffic classifiers are among the ones that have better accuracy [11]. Furthermore, since SVMs represent the fundamental block of many traffic classification techniques, our results can be easily used to extend our analysis to other SVM-based classifiers.

Although this paper focuses on computational costs, we selected classifiers that have similar classification accuracy. In fact, both classifiers have been validated in previous works [2], [14], and resulted equivalent in terms of classification accuracy with more than 90% of traffic (in bytes) correctly classified.

A. DPI classifier

A DPI classifier relies on the observation that each application protocol uses specific *headers* to initiate and/or control the information transfer, which can be described by a regular expression (the *signature*). When a packet belonging to a session not yet associated to an application protocol is delivered to a DPI classifier, this compares the application data with all the signatures associated to the set of supported protocols. Once a suitable signature has been found, it associates the packet (and the entire TCP/IP session) to the corresponding application protocol and then the signature checking is skipped for all the packets belonging to that session.

Although there are several types of DPI classifiers, the most common categories are *(i) packet based, per-flow state* (in short PBFS) which analyzes data on a packet-by-packet basis, and *(ii) message based, per-flow state* (in short MBFS) that analyzes application-level payload as a unique stream of data, after TCP/IP normalization[1]. While PBFS coud appear to be less precise, [1], [2] suggest that its results are roughly equivalent to the ones returned by a more complex MBFS classifier in the vast majority of cases, and hence it represents a better trade-off between accuracy and complexity. This paper will focus on a PBFS classifier, although enriched with the capability to analyze correlated sessions[2] albeit still on a packet-by-packet base.

B. SVM-based classifier

The SVM classifier considered in this paper requires a preliminary training phase to build a statistical model for each of the p protocols under examination. Each protocol is modeled as Gaussian–like test function that is used to determine the probability that a given observed session has been generated by the protocol. Given an observed traffic session, the value of the test function for protocol j is computed after the session has been transformed into a vector in a d_j-dimensional space, $\mathbf{x} = (x_1, x_2, ...x_{d_j})$. Here x_i represents the feature values extracted from the $i-th$ observed packet, excluding all those packets that do not carry application–level payload. The test value for protocol j is then computed as follows:

$$f_j(\mathbf{x}) = \sum_{n=1}^{s} \alpha_n \, \mathcal{N}(\mathbf{x}|\mathbf{y}_{SVn}, \sigma) \tag{1}$$

where the number s of Gaussian terms $\mathcal{N}$, their weights α_n and centers $\mathbf{y}_{SVn}$ (i.e., the *Support Vectors*) and the standard deviation σ, common to all terms, are determined during the training phase by analyzing a reasonable (on the order of hundreds) number of sessions generated by this protocol.

As d_j depends on the protocol, the classification of an observed session should be delayed until all the test functions can be computed, i.e., when at least $N_{SVM} = \max(d_j)$, $(1 \leq j \leq p)$ packets carrying payload have been observed, where this value is determined in the training phase and it is usually on the order of four/five packets. In this case the classifier computes the decision function for each of the p protocols and assigns the observed session to the protocol whose value $f_j(\mathbf{x})$ is largest, provided that it is above a threshold calculated during the training phase. In case all the test values are below the corresponding thresholds, the observed session is declared *unknown*.

IV. Modeling the classifiers

This Section presents a model of the classifiers under examination, which is based on the operations that are performed by each classifier when a packet has to be examined.

A. General behavior of traffic classifiers

Traditional traffic classifiers operate on *sessions*, i.e. a bi-directional ordered sequence of packets exchanged between two hosts and identified by the 5-tuple: end-point IP addresses, transport-layer ports and transport protocol.

Each traffic classification algorithm can be divided into a *slow path* and a *fast path*. The slow path identifies the portion of the classifier that handles packets belonging to unknown

[1]TCP/IP normalization is a common term used to include the set of techniques that are able to cope with fragmented IP packets (returning the original re-assembled IP datagram) and that are able to re-create the TCP stream starting from individual TCP segments.

[2]Some protocols (e.g., FTP or SIP) define a control connection that is also used to negotiate the network parameters of a following data transfer, which will occur in a TCP/IP separate session. We usually refer to the latters as "correlated sessions".

sessions and uses an algorithm that is specific of each classifier (e.g. regular expression matching for DPI). The slow path relies on the result of the previous step to associate following packets (belonging to same session) to the correct protocol and it is equivalent in all the classifiers.

When a new session is classified, a new entry is created in a data structure commonly named *Session Table*, which contains the 5-tuple, the application-level protocol, and a timestamp keeping the time of the last packet (belonging to that session) as seen by the classifier. The fast path has mainly to update the timestamp associated to the current flow in the session table, which is used to delete inactive sessions from the above table in case the session termination cannot be detected. This is rather common especially in case of UDP flows; in this case, a timeout of 10 minutes [23] is usually considered.

B. General cost model

In order to determine the computational cost of each classifier, we modelled each algorithm through decision blocks (diamond-shaped in Figures 1 and 2), which are supposed to be executed at no cost[3], and processing blocks (rectangular blocks) executed at a cost c_i. The overall result is presented in Figures 1 (DPI classifier) and 2 (SVM classifier).

The worst case cost c_{max} experienced by a packet entering the classifier is represented by the cost of executing the entire slow path[4], which is determined once the costs of all the composing blocks are known. In a general scenario, however, the probability that all packets enter the slow path is very low and therefore the average cost is a better indication of the actual cost of each algorithm. In this respect, we introduce a technique to determine the average cost per packet of each classifier on a given network trace in order to compare their computational efficiency on a particular traffic mix.

Given a packet at decision block j we first estimate the transition probabilities towards the two out–paths, respectively $\sigma_{j,i}$ the probability of a transition to block i and $1 - \sigma_{j,i}$ to the other. The transition probability $\sigma_{j,i}$ is important because it can be used to determine exactly which and how a path (and each block within it) is executed. Since a packet can get to block i through different paths (paths are all the possible traversal of the graph from the beginning to the ending node), the global probability to execute block i will be the sum of the probabilities to reach the given block on all the possible paths Q that include both nodes j and i:

$$p_i = \sum_{w=1}^{Q} \prod_{j=1;j,i \in Q_w} \sigma_{j,i} \tag{2}$$

Given these hypothesis and being N the number of processing blocks in each model, we define the average cost per packet $\overline{c}$ as:

$$\overline{c} = \sum_{i=1}^{N} p_i \cdot c_i \tag{3}$$

Both p_i and c_i cannot be determined analytically as they depend on other parameters: for instance different traffic mixes can lead to very different values of p_i. The computational cost c_i is even more difficult to capture because it depends on some pre–calculated constants (e.g., the number of the protocols we want to classify, the number of support vectors) or on some data available only at run-time (e.g., the size of the incoming packet, the number of packets already examined in the session). All these parameters will be presented in Section V and their values will be derived in Section VI based on the observation of real traffic dynamics and by choosing the appropriate running configuration for our classifiers.

The choice of building a model and then measuring its parameters instead of measuring directly the performance of each algorithm is due to two factors. First, this allows to decompose the algorithms in their main functions, making easier the comparison and finding the common parts. Second, since the cost of the algorithms heavily depends on the traffic mix (as shown in Section VI-D), the model provides a better way to extend these results to different traffic traces either by re-measuring the characterizing parameters, or by simulating the results achievable with different values.

C. Modelling of the processing blocks

This Section presents an high-level view of the functions performed by each block. Since some of the processing blocks presented below change their cost depending on parameters that are known only at run-time, in this simple model we characterize their behavior with the worst-case processing cost. This simplification has been evaluated experimentally and we verified that the difference between the actual cost and the one used for modelling is negligible.

The **SessionID Extraction** block (*DPI, SVM*) extracts the SessionID parameters (i.e., IP addresses, transport-level protocol, port numbers) to be used for session lookup. This block and the following are always executed when a new packet enters in the system (i.e., $p_i = 1$).

The **Session lookup** block (*DPI, SVM*) checks if the session the packet belongs to is already present in the Session Table; in this case it updates also the timestamp associated to the current session.

The **Correlated Session** block (*DPI*) analyzes the session-level payload and looks for data that may lead to the identification of a correlated session. This block is executed only in case the application-level protocol may originate correlated sessions (e.g., SIP or FTP). In case the information about a correlated session is found (e.g., the PORT or the PASV commands of an FTP control session), it inserts the 5-tuple of the new correlated session in the Session Table.

The **Pattern matching** block (*DPI*) implements the pattern matching algorithm that looks for the presence of a signature in the application payload. If a signature matches, the Session

[3]Although this is formally incorrect, experimental measurements confirm that the cost of the operations carried out in decision blocks is negligible compared to the one of processing blocks. Further refinements of the model are left to future work.

[4]The slow path is represented by the blocks that are aligned strictly below the starting block.

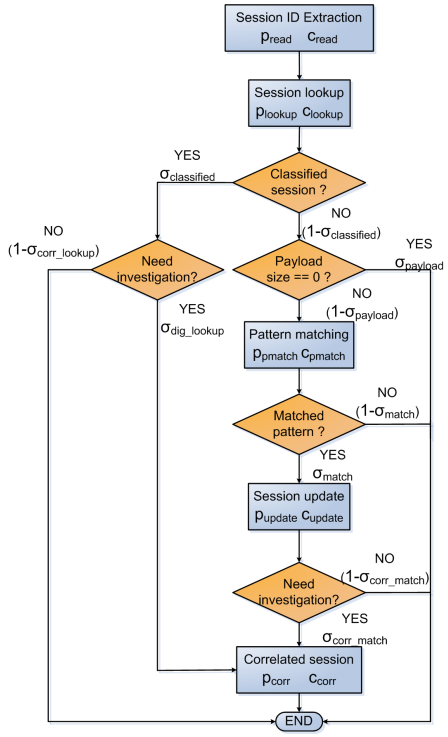

Figure 1. Model of the DPI classifier.

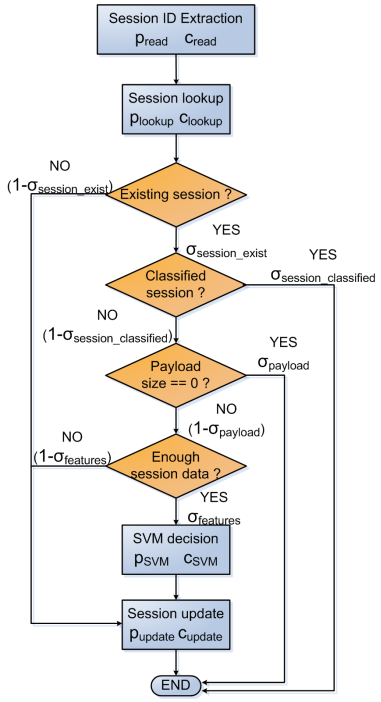

Figure 2. Model of the SVM-based classifier.

Update block adds the new entry to the Session Table. This block is executed on all the packets belonging to the current session until a match is found or the session terminates.

The **SVM decision** block (*SVM*) implements the SVM algorithm that returns the application-layer protocol according to the values of the features extracted from the first packets of the session. The features the classifier uses are the size and the direction (from the originator to the called party or vice versa) of the first payloads transmitted in a session. This block evaluates p times Equation 1[5] and determines the correct protocol (if any) by selecting the function that returned the highest mark. This block is executed once per session when enough packets (N_{SVM}) carrying application-level payload have been seen. After emitting its verdict, SVM does not inspect any following packet even if the protocol is unknown.

The **Session Update** block (*DPI, SVM*) has different behaviors in the two classifiers. The SVM classifier uses this block to store the classification verdict for a session or, in case the number of packets transmitted for a session is still not sufficient (i.e., less than N_{SVM}), it is used to store the values of the current observed features. Instead, the DPI classifier executes this block only when the Pattern Matching block returns a positive match, in which case it stores the outcome of the classification in the Session Table.

D. Cost of the DPI classifier

The model of the DPI classifier includes 5 processing blocks organized as shown in Figure 1. Its cost can be expressed by the following Equation:

$$c_{\text{DPI}} = c_{\text{read}} + c_{\text{lookup}} + p_{\text{pmatch}} \cdot c_{\text{pmatch}} + p_{\text{update}} \cdot c_{\text{update}} + p_{\text{corr}} \cdot c_{\text{corr}} \quad (4)$$

The most expensive portion of the formula is represented by c_{pmatch} that returns the cost of the pattern matching block. Its contribution is mitigated by the weight p_{pmatch} that represents the probability that a packet belonging to a unclassified flow is fed into the classifier. Briefly, the average classification cost per packet is inversely proportional to the length of the sessions (because the term p_{pmatch} tends to zero). Vice versa, the cost will increase in presence of unclassifiable sessions (e.g. encrypted sessions, which are not identifiable by a DPI classifier) because in that case the p_{pmatch} value will become closer to 1.

E. Cost of the statistical classifier

The model of the SVM classifier includes 4 processing blocks organized as shown in Figure 2. Its cost can be expressed by the following Equation:

$$c_{\text{STAT}} = c_{\text{read}} + c_{\text{lookup}} + p_{\text{SVM}} \cdot c_{\text{SVM}} + p_{\text{update}} \cdot c_{\text{update}} \quad (5)$$

The SVM Decision block is executed once per session, when the classifier receives enough packets for identifying the application protocol. As for the DPI case, the cost of this classifier decreases when the average session length is high. Note that for short sessions (i.e., sessions that have less than N_{SVM} packets with application-level payload) the classifier never executes the SVM decision block.

V. Implementation of the Classifiers

This Section presents the implementation of our classifiers and derives the cost of each block, which depends on the characteristics of the model and the inputs provided.

[5]It is worthy remembering that p represents the number of protocols that the classifier has been trained on.

The **SessionID Extraction** block has been implemented using the code generated by the NetVM framework [19], which includes a Just-In-Time compiler that generates a protocol parser in native assembly code for the x86 architecture.

The **Session Lookup** and **Session Update** blocks have been implemented using the *Hash_map* container of C++ Extended STL library [18]. This guarantees a $O(1)$ complexity in the average usage, supposing that the hash function distributes entries uniformly over the hash table. Under these assumptions (which will be verified in Section VI through direct measurements), the cost will be constant.

The implementation of the **Correlated Sessions** block strictly depends on the header format of the protocols we are considering. Since this block currently operates on text-based protocol such as SIP and FTP, we implemented the C++ code that locates the fields related to session correlation through appropriate regular expressions. Due to the small number of executions of this block and the limited difference in terms of processing costs with respect to different protocols, we decided to model the complexity of this block as $O(1)$, corresponding to the worst–case execution cost.

The most important block for the DPI classifier is the **Pattern matching** one. We chose Deterministic Finite Automata (DFA) because their cost has a linear dependence on the number of input characters (which is bounded by the MTU size in case of a PBFS classifier) and it does not depend on the number of regular expressions. Its drawback is that the automaton that represents the set of regular expressions may require large amount of memory, depending on the characteristics of the regular expressions.

As shown in Figure 3, the memory occupied by the DFA increases linearly when the input patterns contains only *(i) anchored regexp* (i.e., begins with the '^' sign), that identifies the regular expressions whose pattern must be found at the beginning of the payload (first region on the left). The slope increases when we add also *(ii) not anchored regexp not containing the Kleen closure* (i.e. the '*' wildcard), in which the regular expression can be found in any point of the input data (second region). Finally, the memory tends to explode when we add the second *(iii) not anchored regexp containing Kleen closure*, due to the possible ambiguities in the input pattern that force the addition of a large number of states for matching all the possible cases. It is worthy noting that the number of states explodes when at least *two* expressions of type *(iii)* are merged together.

Althoug DFA state explosion has been widely investigated and several solutions exist (e.g., [20]–[22]) we opted for the simpler DFA after an analysis of our classification patterns, which were based on the regular expressions defined in the NetBee library [17]. Among these patterns, only two were of type *(iii)*, the first encapsulated in TCP while the other in UDP. Since we created two different DFAs for TCP and UDP-based protocols, we did not experience any memory explosion and the total amount of memory used was about 3MBytes, roughly split in half between TCP and UDP, which is a reasonable value even for embedded implementations. While

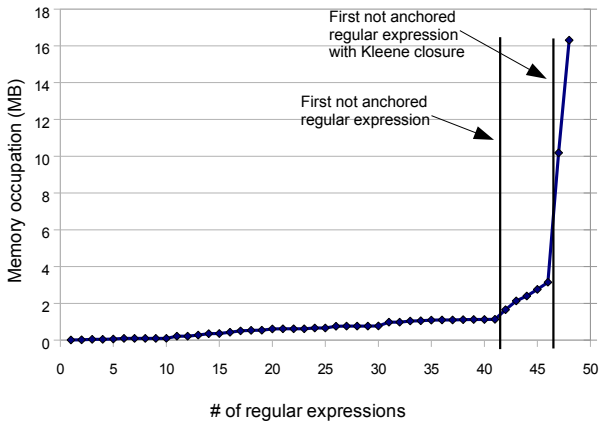

Figure 3. DFA memory occupation.

leaving further analysis in terms of memory requirements for future work, this result confirms that our implementation of the DPI algorithm is viable, considering that the pattern matching block represents the larger contributor to the memory requirements of the DPI.

The code for pattern matching has been generated using Flex, a well known tool for generating lexical scanners, which was able to create pure DFAs (without any state compression) through the appropriate configuration. As stated by the theory, the cost of this block is linearly proportional to the number of the input characters read by the automata, according to the following Equation:

$$c_{\text{pmatch}} = k \cdot S + u \tag{6}$$

where S is the number of input characters, k is a linear coefficient and u is a costant offset due to the initialization of the DFA.

The **SVM decision** block computes Equation 1 for each protocol j, which consists in a sum of d_j-dimensional Gaussians, each one centered in one of the s_j Support Vectors. The computational complexity of this block depends on the parameters of the protocol models: the dimension d_j (i.e, the number of features), the number of Support Vectors s_j (i.e., the number of Gaussians) and the overall number of protocols under examination p. The values for d_j and s_j are specific of the j-th protocol and are determined in the training phase.

To optimize the computational cost of the SVM test functions, we decided to use a basic property of the multivariate Gaussian joint density function that allows to factorize it as a product of marginal densities. Given this premise, we precomputed only one marginal function for each protocol and stored it in memory: this was possible as we have the same standard deviation along all dimensions. During classification, we multiply the d_j values of each marginal of the s_j Gaussians and sum the s_j achieved values. Therefore, for each session $\mathbf{x}$ we execute the steps we show in Figure 4.

We associate the coefficients $(\underline{a}, \underline{b}, \underline{c})$ to the operations in Figure 4, which determine the computational cost for each

```
1:  for j = 1 : p
2:    for i = 1 : s_j
3:      for k = 1 : d_j
4:        a: read the value of the k-th
5:        marginal of the i Gaussian
6:        and multiply for previous
7:      end
8:      b: sum to values of the other
9:      Guassians in the point x
10:   end
11:   c: find the max of the computed values
12:   (i.e. the candidate protocol) that is
13:   above the corresponding threshold
14: end
```

Figure 4. Sequence of steps to express of the verdict in the SVM classifier.

Table I
TRACES USED IN OUR EXPERIMENTS.

Data set	Date and Duration	Volume	Packets
UNIBS	April 2 and 4, 2008 5 + 6 hours	36.6 + 35.7 GB 96.8% + 98.3% TCP	265M 94.3% TCP
POLITO	Dec. 20, 2007 12 hours	419 GB 94.7 % TCP	579M 92.3% TCP

dimension and for each Support Vector of the j-th protocol. Using these coefficients we derive the computational cost for the execution of this block as:

$$c_{\text{SVM}} = \sum_{j=1}^{p} \left[(\underline{\text{a}} \cdot d_j + \underline{\text{b}}) \cdot s_j + \underline{\text{c}} \right] \ . \tag{7}$$

We compute the decision function value for each of the p protocols in the point **x** in which the current session is located, therefore the complexity of the block depends linearly on the number of protocols under examination (i.e., on the number of functions to be evaluated).

VI. EXPERIMENTAL ANALYSIS

This Section is dedicated to the experimental analysis of our classifiers, which is based on a set of real traces used to derive the parameters of our model. The overall result will be an indication on the performance of the algorithms under evaluation in a real deployment scenario.

A. Traffic traces

This paper exploits two traffic traces (whose main characteristics are shown in Table I) that were collected through the well-known *tcpdump* tool at the border routers of our Universities, and are hence called UNIBS and POLITO data sets. Due to the necessity to inspect the payload (for DPI), we were unable to use public traces which maintain only a portion of the application-level data for privacy reasons.

Traces, which include only TCP and UDP traffic, have different traffic characteristics: while the POLITO data set resembles traditional enterprise traffic (e.g., P2P applications are limited), the UNIBS data set contains a large portion of peer-to-peer traffic that is known to stress DPI classifiers due to the use of hiding techniques.

After inspecting the traces, we decided to train the SVM classifier to analyze only protocols responsible for the largest share of traffic, primarily due to the necessity to have enough sessions for the training, while the DPI classifier was able to recognize 48 different protocols. The training phase was done using the first portion of each trace and used a DPI classifier (in addition to manual inspection) to derive the application protocol associated to each session and to build the SVM data. Table II lists the protocols selected in our experiments: all of them included at least 1000 sessions in the training data set.

Table II
PROTOCOLS ANALYZED BY THE SVM CLASSIFIER.

UNIBS (tcp)	UNIBS (udp)	POLITO (tcp)	POLITO (udp)
bittorrent edonkey http, msn pop3, smtp ssl	bittorrent dns edonkey skype	bittorrent edonkey http, msn pop3, smtp imap, smb ssl, ssh	bittorrent dns edonkey skype

Table III
EXECUTION PROBABILITIES p_i AND TRANSITION PROBABILITIES σ_j.

Execution freq.	UNIBS-tcp	UNIBS-udp	POLITO-tcp	POLITO-udp
SVM				
p_{SVM}	0.003	0.004	0.007	0.027
p_{update}	0.030	0.196	0.054	0.310
$\sigma_{\text{session\_exists}}$	0.99	0.90	0.98	0.86
$\sigma_{\text{session\_classified}}$	0.94	0.89	0.81	0.79
σ_{payload}	0.61	0.00	0.79	0.00
σ_{features}	0.13	0.04	0.19	0.15
DPI				
p_{pmatch}	0.067	0.100	0.056	0.200
p_{corr}	17e-5	10e-5	41e-5	66e-6
p_{update}	0.004	0.070	0.011	0.072
σ_{session}	0.88	0.90	0.84	0.80
$\sigma_{\text{corr\_lookup}}$	19e-5	11e-5	48e-5	82e-6
σ_{payload}	0.44	0.00	0.65	0.00
σ_{match}	0.06	0.70	0.20	0.36
$\sigma_{\text{corr\_match}}$	26e-5	90e-7	73e-5	23e-7

B. Execution probability of each block

The execution probability p_i can be easily derived from the transition probability σ_j through Equation 2. As shown in Table III, transition probabilities are fairly different in the two traces due to the different traffic mix (e.g., the duration of the sessions, the number of packets without application-level payload, etc.). Particularly, the UNIBS data set is more challenging for the DPI classifier since a fair amount of traffic is encrypted and hence never classified, forcing several packets to pass through the slow path anyway.

C. Computational time of each block

In this Section we evaluate the computational cost of each basic block. Since the execution time of these blocks is rather small, we counted clock ticks through the RDTSC assembly instruction available on Pentium-compatible processors.

Measurements were done by profiling the worst-case execution path of each block, evaluated with real traffic. Measurements were repeated 1M times and the average cost was derived by discarding the top and bottom 10% results. In order

Table IV
COST OF EACH BASIC BLOCK IN CLOCK TICKS.

Block name	UNIBS-tcp	UNIBS-udp	POLITO-tcp	POLITO-udp
Sess. ID Extractor	78	78	78	78
Sess. Lookup/Upd.	49	49	49	49
Correl. Sessions	1850	1850	1850	1850
SVM Decision	26865	24176	42915	20979
Pattern Match	13308	1219	9051	2988

to mitigate the effects of context switching and cache filling, measurements were executed with no other jobs running on the machine apart from essentials system daemons. The measurement platform was an Intel Dual Xeon 5160 at 3GHz, 4GB RAM and Ubuntu 8.04 32bit; the code under examination was compiled with GCC v4.2.4 with -O3 optimization level and always executed on the first core.

The *SessionID Extractor* block returned an average value of 78 clock ticks for parsing a single packet up to the transport layer using our traffic traces.

Since both *Session Lookup* and *Session Update* are implemented with the same piece of code, these blocks have equal costs. Particularly, the cost has been simulated by implementing a table up to 5M valid entries (although we never exceeded 100K entries in our traces) and measuring the ticks required to search and update a single random entry. Results confirm that in practical cases this implementation does not depend on the number of entries present in the hash table, although in theory the worst–case depends on the necessity to handle collisions on the same key. This means that the hash function is well balanced and the hash table is filled uniformly. The worst-case measured cost associated to these blocks is 49 ticks.

The *Correlated Sessions* block has been evaluated by measuring the cost of extracting the information about correlated sessions in case of SIP and FTP (control) packets; although their cost is slightly different, we took the worst-case cost which is 1850 ticks.

The real distribution of the cost of the *Pattern Matching* block is shown in Figure 5 (with respect to the POLITO traffic trace) and the upper bound is described by Equation 6 with parameters $k = 12.5$ and $u = 472.7$ (derived experimentally). In fact, the actual cost is often smaller than the value predicted by this function because of the large number of regular expressions with the '^' anchor, which interrupt the exploration of the DFA as soon as an accepting state is reached, thus terminating the processing before the end of payload. The worst–case cost for the pattern matching is 18650 ticks; values in Table IV are referred to the cost in presence of the average packet size, as derived from our traffic traces.

The cost of the *SVM Decision Block* depends on the output of the training phase: given a protocol j we have a number of Support Vectors s_j and packets to evaluate d_j, that are determined from the characteristics of the training observations. In order to derive the numbers presented in Table IV, we had to to estimate the coefficients $(\underline{a}, \underline{b}, \underline{c})$ present in Equation 7. These coefficients are independent from the traces used and

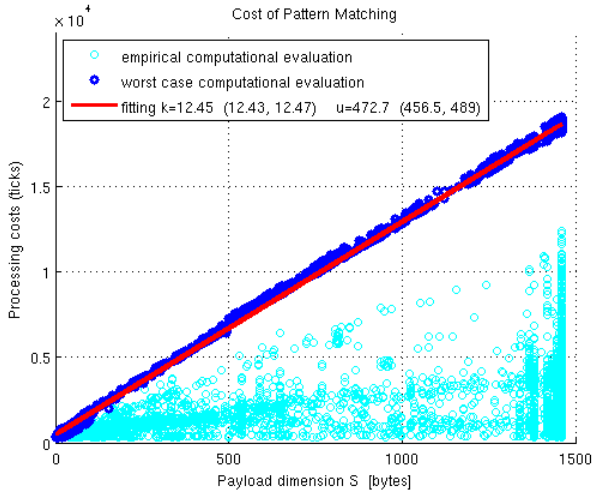

Figure 5. Linear least squares fitting of the measures of the computational costs of the pattern matching as function of the payload size S.

allow us to obtain the cost c_{SVM} once s_j, d_j, p have been determined in the training phase. In order to determine the value of these three coefficients, we executed a sequence of measures of the complexity c_{SVM} with different values of s_j, d_j, p and obtained a set of linear functions of c_{SVM} in one of the parameters. Finally, we estimated the linear least squares fitting of the achieved measurements and we obtained values $\underline{a} = 4.3$, $\underline{b} = 14.5$ and $\underline{c} = 8.0$ for the three coefficients.

Table IV summarizes the cost of each block with the runtime parameters defined in our systems.

D. Overall computational time estimation

Using the Equations derived in Section IV, the transition probabilities and the costs of the processing blocks measured in this Section, we calculate the cost for classifying TCP and UDP traffic with both classifiers; results are shown in Figure 6. The worst case represents the cost of the slow path, the best case refers to the fast path, while the average line represents the total cost of processing our traces over the number of packets. Results show that the average cost of the two classifiers is similar, and the same holds for worst and best cases.

Particularly, the DPI classifier performs worse on the UNIBS trace in the average case because of the large percentage of (often encrypted) P2P traffic over TCP, which forces the DPI classifier to analyze all packets of these sessions because no matching signatures are found. However, even in this case the cost of the DPI performances are comparable to SVM. The computational complexity of the SVM-based classifier is larger for the POLITO data set than UNIBS, mostly due to the different number of protocols we use for the two data sets (10 vs. 7 TCP-based protocols, plus 4 UDP-based ones in both traces); however, the average case is comparable to the cost of the DPI classifier (although higher than in UNIBS traces). It is worty noting that these results are biased toward SVM because of the different number of protocols supported

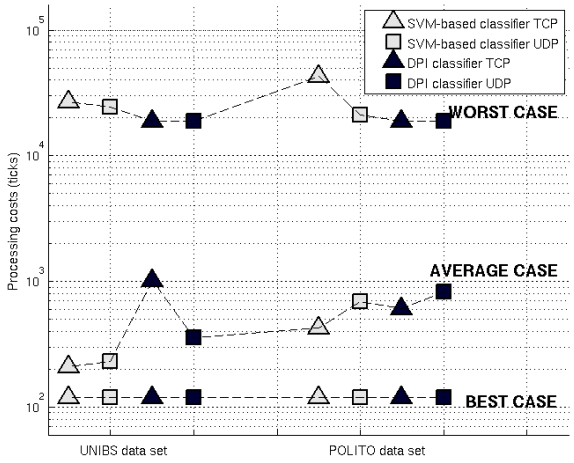

Figure 6. Overall computational time estimation for each packet estimated for the two traffic traces.

by SVM (respectively 11 and 14) compared to DPI (48), as shown in Section VI-A. Since the processing cost of SVM is directly proportional to the number of protocols, increasing this number in SVM could lead to different results that may privilege the DPI.

As expected, the best case is the same for both classifiers, while no much difference exists between TCP and UDP traffic.

VII. Conclusions and future work

In this paper we presented a preliminary analysis of the computational complexity of a DPI-based classifier. The DPI approach exhibits costs that are comparable to the ones of an SVM-based classifier: depending on the traffic trace's composition (in terms of the types of protocols that are present), the DPI classifier can be as much computationally complex as the SVM-based one in the average, or can show a complexity that goes as high as five times the one of the SVM classifier (especially in case SVM recognizes a limited number of protocols). In all cases, the differences are not as dramatic as we previously thought, and warrant a reconsideration of the type of usage that DPI techniques should have in traffic classification.

Although interesting, our results are still preliminary, and further work is required to generalize them. First of all, we are working to include memory costs in the analysis, studying how the trade-off between computational and memory costs can affect the comparison between statistical and DPI classifiers. Second, we are extending our analysis to other statistical classifiers, such as the ones based on Naïve-Bayes approaches [7] and Gaussian Mixture Models [8]. Third, although the two approaches compared in this paper show similar overall accuracy [2], [14], further work is needed to correlate precisely complexity and accuracy. Finally, the analysis must be extended to other traffic traces and other scenarios before the validity of our results can be generalized.

References

[1] A. Moore, K. Papagiannaki, Toward the Accurate Identification of Network Application, 6th International Workshop on Passive and Active Network Measurement, Boston MA, USA, May 2005, pp. 41-54.

[2] F. Risso, A. Baldini, M. Baldi, P. Monclus, O. Morandi, Lightweight, Payload-Based Traffic Classification: An Experimental Evaluation, IEEE International Conference on Communications (ICC 2008), Beijing (China), pp. 5869-5875, May 2008.

[3] J. Erman, A. Mahanti, M. Arlitt, C. Williamson, Identifying and discriminating between web and peer-to-peer traffic in the network core, Proceedings of the 16th International Conference on World Wide Web, Banff, Alberta, Canada pp. 883 - 892, 2007.

[4] J. Erman, M. Arlitt, A. Mahanti, Traffic classification using clustering algorithms, Proceedings of the 2006 SIGCOMM, Pisa, Italy, pp. 281 - 286, 2006.

[5] L. Bernaille, R. Teixeira, I. Akodkenou, Traffic classification on the fly, 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems, San Jose, CA, pp. 40-49, 2008.

[6] S. Zander, T. Nguyen, G. Armitage, Self-learning IP traffic classification based on statistical flow characteristics, International Workshop on Passive and Active Network Measurement, Boston MA, pp. 325-328, 2005.

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

[8] L. Bernaille, R. Teixeira, K. Salamatian, Early Application Identification, 2nd CoNEXT Conference, Lisboa, Portugal, Dec. 2006.

[9] A. Este, F. Gringoli, L. Salgarelli, Support Vector Machines for TCP traffic classification, Elsevier Computer Network, 53(14), pp. 2476 - 2490, 2009.

[10] N. Williams and S. Zander and G. Armitage, A Preliminary Performance Comparison of Five Machine Learning Algorithms for Practical IP Traffic Flow Classification, SIGCOMM Computer Communication Review, Vol. 36, No. 5, , pp. 7-15, Oct. 2006.

[11] H. Kim, Kc Claffy, M. Fomenkova, D. Barman and M. Faloutsos, Internet Traffic Classification Demystified: The Myths, Caveats and Best Practices, ACM CoNEXT, Madrid, Spain, Dec. 2008.

[12] WEKA, http://www.cs.waikato.ac.nz/ml/weka .

[13] T. Karagiannis, K, Papagiannaki, M. Faloutsos, BLINC: Multilevel traffic classification in the Dark, ACM SIGCOMM, Aug. 2005.

[14] A. Este, F. Gargiulo, F. Gringoli, L. Salgarelli, C. Sansone, Pattern Recognition Approaches for Classifying IP Flows, 7th International Workshop on Statistical Pattern Recognition, Orlando, FL, Dec. 2008.

[15] V.N. Vapnik, Statistical Learning Theory. John Wiley and Sons, New York, 1998.

[16] B. Schölkopf, J.C. Platt, J. Shawe–Taylor, A.J. Smola, R.C. Williamson, Estimating the Support of a High–Dimensional Distribution. Neural Computation, **13**, pp. 1443–1471, 2001.

[17] Computer Networks Group (NetGroup) at Politecnico di Torino. The NetBee Library. August 2004. [online] Available at http://www.nbee.org/.

[18] Hash_map container reference, http://www.sgi.com/tech/stl/hash_map.html

[19] O. Morandi, F. Risso, M. Baldi, A. Baldini, Enabling flexible protocol processing through dynamic code generation, International Conference on Communications, Beijing (China), pp. 5849 - 5856, May 2008.

[20] R. Smith, C. Estan, S. Jha, S. Kong, Deflating the big bang: fast and scalable deep packet inspection with extended finite automata, ACM SIGCOMM Computer Communication Review, Volume 38, Issue 4 (October 2008), pp. 207-218.

[21] M. Becchi, P. Crowley, Efficient regular expression evaluation: Theory to pratice, Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems, San Jose, CA, pp. 50-59, 2008.

[22] S. Kumar, S. Dharmapurikar, F. Yu, P. Crowley, J. Turner, Algorithms to accelerate multiple regular expressions matching for deep packet inspetion, ACM SIGCOMM Computer Communication Review, Volume 36, Issue 4, pp. 339 - 350, October 2006.

[23] N. Brownlee, Traffic flow measurement: Meter MIB, Request for Comments RFC 2064, Internet Engineering Task Force, January 1997.