Kiss to Abacus: a comparison of P2P-TV traffic classifiers

Alessandro Finamore[1], Michela Meo[1], Dario Rossi[2], Silvio Valenti[2]

[1] Politecnico di Torino, Italy
[2] TELECOM Paristech, France

Abstract. In the last few years the research community has proposed several techniques for network traffic classification. While the performance of these methods is promising especially for specific classes of traffic and particular network conditions, the lack of accurate comparisons among them makes it difficult to choose between them and find the most suitable technique for given needs.
Motivated also by the increase of P2P-TV traffic, this work compares Abacus, a novel behavioral classification algorithm specific for P2P-TV traffic, and Kiss, an extremely accurate statistical payload-based classifier. We first evaluate their performance on a common set of traces and later we analyze their requirements in terms of both memory occupation and CPU consumption. Our results show that the behavioral classifier can be as accurate as the payload-based with also a substantial gain in terms of computational cost, although it can deal only with a very specific type of traffic.

1 Introduction

In the last years, Internet traffic classification has attracted a lot of attention from the research community. This interest is motivated mainly by two reasons. First, an accurate traffic classification allows network operators to perform many fundamental activities, e.g. network provisioning, traffic shaping, QoS and lawful interception. Second, traditional classification methods, which rely on either well-known ports or packet payload inspection, have become unable to cope with modern applications (e.g., peer-to-peer) or with the increasing speed of modern networks [1, 2].

Researchers have proposed many innovative techniques to address this problem. Most of them exploit statistical properties of the traffic generated by different applications at the flow or host level. These novel methods have the advantage of requiring less resources while still being able to identify applications which do not use well-known ports or exploit encrypted/closed protocols. However the lack of accurate and detailed comparisons discourages their adoption. In fact, since each study tests its own algorithm on a different set of traces, under different conditions and often using different metrics, it is really difficult for a network operator to identify which methods could best fit its needs.

In this paper, we face this problem by comparing two traffic classifiers, one of which is specifically targeted to P2P-TV traffic. These applications, which are rapidly gaining a very large audience, are characterized by a P2P infrastructure providing a live streaming video service. As next generation of P2P-TV services are beginning to offer HD

content, the volume of traffic they generate is expected to grow even further, so that their identification is particularly interesting for network operators. The two considered classifiers are the first ones to handle this kind of traffic and exploit original and quite orthogonal approaches. Kiss [3] is a statistical payload-based classifier and it bases the classification on the examination of the first bytes of the application-layer payload. It has already been compared with other classifiers in [4], proving to be the best one for this specific class of traffic. Abacus [5], instead, is a behavioral classifier, which derives a statistical representation of the traffic patterns generated by a host by simply counting the number of packets and bytes exchanged with other peers during small-time windows. This simple approach can capture several distinctive properties of different applications, allowing their classification.

We test the two techniques on an common set of traces, evaluating their accuracy in terms of both true positives (i.e., correct classification of P2P-TV traffic) and true negatives (i.e., correct identification of traffic other than P2P-TV). We also provide a detailed comparison of theirs features, focusing mostly on the differences which stem from the undertaken approaches. Moreover, we formally investigate the computational complexity by comparing the memory occupation and the computational costs.

Results show that Abacus achieves practically the same performance of Kiss and both classifiers exceed 99% of correctly classified bytes for P2P-TV traffic. Abacus exhibits some problems in terms of flow accuracy for one specific application, for which it still has a high bytewise accuracy. The two algorithms are also very effective when dealing with non P2P-TV traffic, raising a negligible number of false negatives. Finally we found that Abacus outperforms Kiss in terms of computation complexity, while Kiss is a much more general classifier, able to work with a wider range of protocols and network conditions.

The paper is organized as follows. In Sec. 2 we present some work related to ours. In Sec. 3 we briefly present the two techniques under exam, then in Sec. 4 we test them on a common set of traces and compare their performance. We proceed with a more qualitative comparison of the classifiers in Sec. 5 as well as an evaluation of their computational cost. Finally Sec. 6 concludes the paper.

2 Related Work

Recently, many works have been focusing on the problem of traffic classification. In fact, traditional techniques like port-based classification or deep packet inspection appear more and more inadequate to deal with modern networks and applications [1, 2]. Therefore the research community has proposed a rather large number of innovative solutions, which consist notably in several statistical flow-based approaches [6–8] and in a fewer host-based behavioral techniques [9, 10].

The heterogeneity of these approaches, the lack of a common dataset and the lack of a widely approved methodology make a fair and comprehensive comparison of these methods a daunting task [11]. In fact, to date, most of the comparison effort has addressed the investigation of different machine learning techniques [6–8], using the same set of features and the same set of traces.

More recently, a few works have specifically taken into account the comparison problem [12–14]. The authors of [12] present a qualitative overview of several machine learning based classification algorithms. On the other hand, in [13] the authors compare three different approaches (i.e., based on signature, flow statistics and host behavior) on the same set of traces, highlighting both advantages and limitations of the examined methods. A similar study is carried also in [14], where authors evaluate spatial and temporal portability of a port-based, a DPI and a flow-based classifier.

The work presented in this paper follows the same direction of these comparative studies, but focuses only on P2P-TV applications. In fact, P2P-TV has been attracting many users in the last few years, and consequently also much consideration from the research community. Moreover, works on this topic consist mainly in measurement studies of P2P-TV application performance in real networks [15, 16]. The two classifiers compared are the only ones proven to correctly identify this type of traffic. Kiss was already contrasted with a DPI and a flow-based classification algorithm in [4], proving itself the most accurate for this class of traffic. Moreover in our study we also take into account the computational cost and memory occupation of the algorithms under comparison.

3 Classification algorithms

This section briefly introduces the two classifiers. Here we focus our attention on the most relevant aspects in a comparison perspective, while we refer the interested reader to [5] and [3] for further details and discussion on parameters settings.

Both Kiss and Abacus employ supervised machine learning as their decision process, in particular Support Vector Machine - SVM [17], which has already been proved particularly suited for traffic classification [13]. In the SVM context, entities to be classified are described by an ordered set of *features*, which can be interpreted as coordinates of points in a multidimensional space. Kiss and Abacus differ for the choice of the features. The SVM must be trained with a set of previously labeled points, commonly referred to as the *training set*. During the training phase, the SVM basically defines a mapping between the original feature space and a new space, usually characterized by an higher dimensionality, where the training points could be separated by hyperplanes. In this way, the target space is subdivided in areas, each associated to a specific class. During the classification phase, a point can be classified simply looking for the region which best fits it.

Before proceeding with the description of the classifiers, it is worth analyzing their common assumption. First of all, they both classify *endpoints*, i.e., couples (IP address, transport-layer port) on which a given application is running. Second, they currently work only on UDP traffic, since this is the transport-layer protocol generally chosen by P2P-TV applications. Finally, given that they rely on a machine learning process, they follow a similar procedure to perform the classification. As a first step, the engines derive a signature vector from the analysis of the traffic relative to the endpoint they are classifying. Then, they feed the vector to the trained SVM, which in turn gives the classification result. Once an endpoint has been identified, all the flows which have

that endpoint as source or destination are labeled as being generated by the identified application.

3.1 Abacus

A preliminary knowledge of the internal mechanisms of P2P-TV applications is needed to fully understand the key idea behind the Abacus classifier. A P2P-TV application performs two different tasks: first, it exchanges video chunks (i.e., small fixed-size pieces of the video stream) with other peers, and, second, it participates to the P2P overlay maintenance. The most important aspect is that it must keep downloading a steady rate of video stream to provide users with a smooth video experience. Consequently, a P2P-TV application maintains a given number of connections with other peers from which it downloads pieces of the video content. Abacus signatures are thus based on the number of contacted peers and the amount of exchanged information among them.

In Tab. 4 we have reported the procedure followed by Abacus to build the signatures. The first step consists in counting the number of packets and bytes received by an endpoint from each peer during a time window of 5 sec. At the beginning, let us focus on the packet counters. We first define a partition of $\mathbb{N}$ in B exponential-sized bins I_i, i.e. $I_0 = [0, 1]$, $I_i = [2^{i-1} + 1, 2^i]$ and $I_B = [2^B, \infty)$. Then, we order the observed peers in bins according to the number of packets they have sent to the given endpoint. In the pseudo-code we see that we can assign a peer to a bin by simply calculating the logarithm of the associated number of packets. We proceed in the same way also for the byte counters (except that we use a different set of bins), finally obtaining two vectors of frequencies, namely p and b. The concatenation of the two vectors is the Abacus signature which is fed to the SVM for the actual decision process.

This simple method highlights the distinct behaviors of the different P2P-TV applications. Indeed, an application which implements an aggressive peer-discovering strategy will receive many single-packet probes, consequently showing large values for low order bins. Conversely, an application which downloads the video stream using chunks of, say, 64 packets will exhibit a large value of the 6-th bin.

Abacus provides a simple mechanism to identify applications which are "unknown" to the SVM (i.e., not present in the training set), which in our case means non P2P-TV applications. Basically, for each class we define a centroid based on the training points, and we label a signature as unknown if its distance from the centiroid of the associated class exceeds a given threshold. To evaluate this distance we use the Bhattacharyya distance, which is specific for probability mass functions. All details on the choice of the threshold, as well as all other parameters can be found in [5].

3.2 Kiss

The Kiss classifier [3] is instead based on a statistical analysis of the packets payload. In particular, it exploits a *Chi-Square* like test to extract statistical features from the first application-layer payload bytes. Considering a window of C segments sent (or received) by an endpoint, the first k bytes of each packet payload are split into G groups of b bits. Then, the empirical distributions O_i of values taken by the G groups over the

Table 1. Datasets used for the comparison

Dataset	Duration	Flows	Bytes	Endpoints
Napa-WUT	180 min	73k	7Gb	25k
Operator 2006 (op06)	45 min	785k	4Gb	135k
Operator 2007 (op07)	30 min	319k	2Gb	114k

C segments are compared to a uniform distribution $E_i = C/2^b$ by means of the Chi-Square like test:

$$X_g = \sum_{i=1}^{2^b} \frac{(O_i^g - E)^2}{E} \quad g \in [1, G] \tag{1}$$

This allows to measure the randomness of each group of bits and to discriminate among constant/random values, counters, etc. as the Chi-Square test assumes different values for each of them. The array of the G Chi-Square values defines the application signature. In this paper, we use the first $k = 12$ bytes of the payload divided into groups of 4 bits (i.e., $G = 24$ features per vector) and $C = 80$ segments to compute each Chi-Square.

The generated signatures are then fed to a multi-class SVM machine, similarly to Abacus. As previously stated, a training set is used to characterize each target class, but for Kiss an additional class must be defined to represent the remaining traffic, i.e., the *unknown* class. In fact, a multi-class SVM machine always assigns a sample to one of the known classes, in particular to the best fitting class found during the decision process. Therefore, in this case a trace containing only traffic other than P2P-TV is needed to characterize the unknown class. We already mentioned that in Abacus this problem is solved by means of a threshold criterion using the distance of a sample from the centroid of the class. We refer the reader to [3] for a detailed discussion about Kiss parameter settings and about the selection of traffic to represent the unknown class in the training set.

4 Experimental Results

4.1 Methodology and Datasets

We evaluate the two classifiers on the traffic generated by four popular P2P-TV applications, namely PPLive, TVAnts, SopCast and Joost [3]. Furthermore we use two distinct sets of traces to asses two different aspects of our classifiers.

The first set was gathered during a large-scale active experiment performed in the context of the Napa-Wine European project [18]. For each application we conduct an hour-long experiment where several machines provided by the project partners run the software and captured the generated traffic. The machines involved were carefully configured in such a way that no other interfering application was running on them, so that

[3] Joost became a web-based application in October 2008. At the time we conducted the experiments, it was providing VoD and live-streaming by means of P2P

Table 2. Classification results

(a) Flows

	Abacus					Kiss						
	pp	tv	sp	jo	un		pp	tv	sp	jo	un	nc
pp	**13.35**	0.32	-	0.06	86.27	pp	**98.8**	-	-	-	0.2	1
tv	0.86	**95.67**	0.15	-	3.32	tv	-	**97.3**	-	0.01	0.69	2
sp	0.33	0.03	**98.04**	0.1	1.5	sp	-	-	**98.82**	-	0.21	0.97
jo	0.06	2.21	-	**81.53**	16.2	jo	-	-	-	**86.37**	3.63	10
op06	0.1	0.1	1.03	0.06	**98.71**	op06	-	0.44	0.08	0.55	**92.68**	6.25
op07	0.21	0.03	0.87	0.05	**98.84**	op07	-	2.13	0.09	1.21	**84.07**	12.5

(b) Bytes

	Abacus					Kiss						
	pp	tv	sp	jo	un		pp	tv	sp	jo	un	nc
pp	**99.33**	-	-	0.11	0.56	pp	**99.97**	-	-	-	0.01	0.02
tv	0.01	**99.95**	-	-	0.04	tv	-	**99.96**	-	-	0.03	0.01
sp	0.01	0.09	**99.85**	0.02	0.03	sp	-	-	**99.98**	-	0.01	0.01
jo		-	-	**99.98**	0.02	jo	-	-	-	**99.98**	0.01	0.01
op06	1.02	-	0.58	0.55	**97.85**	op06	-	0.07	-	0.08	**98.45**	1.4
op07	3.03	-	0.71	0.25	**96.01**	op07	-	0.08	0.74	0.05	**96.26**	2.87

pp=PPLive, tv=Tvants, sp=Sopcast, jo=Joost, un=Unknown, nc=not-classified

the traces contain P2P-TV traffic only. This set is used both to train the classifiers and to evaluate their performance in identifying the different P2P-TV applications.

The second dataset consists of two real-traffic traces collected in 2006 and 2007 on the network of a large Italian ISP. This operator provides its customers with uncontrolled Internet access (i.e., it allows them to run any kind of application, from web browsing to file-sharing), as well as telephony and streaming services over IP. Given the extremely rich set of channels available through the ISP streaming services, customers are not inclined to use P2P-TV applications and actually no such traffic is present in the traces. We verified this by means of a classic DPI classifier as well as by manual inspection of the traces. This set has the purpose of assessing the number of false alarms raised by the classifiers when dealing with non P2P-TV traffic. We report in Tab. 1 the main characteristics of the traces.

To compare the classification results, we employ the `diffinder` tool [19], as already done in [4] . This simple software takes as input the logs from different classifiers with the list of flows and the associated classification outcome. Then, it calculates as output several aggregate metrics, such as the percentage of agreement of the classifiers in terms of both flows and bytes, as well as a detailed list of the differently classified flows, so eventually enabling further analysis.

4.2 Classification results

Tab. 2 reports the accuracy achieved by the two classifiers on the test traces. Each table is organized in a confusion-matrix fashion where rows correspond to real traffic i.e.

the expected outcome, while columns report the possible classification results. For each table, the upper part is related to the Napa-Wine traces while the lower part is dedicated to the operator traces. The values in bold on the main diagonal of the tables express the *recall*, a metric commonly used to evaluate classification performance, defined as the ratio of true positives over the sum of true positives and false negatives. The "unknown" column counts the percentage of traffic which was recognized as not being P2P-TV traffic, while the column "not classified" accounts for the percentage of traffic that Kiss cannot classify as it needs at least 80 packets for any endpoint.

At first glance, both the classifiers are extremely accurate in terms of bytes. For the Napa-Wine traces the percentage of true positives exceeds 99% for all the considered applications. For the operator traces, again the percentage of true negatives exceeds 96% for all traces, with Kiss showing a overall slightly better performance. These results demonstrate that even an extremely lightweight behavioral classification mechanism, such as the one adopted in Abacus, can achieve the same precision of an accurate payload based classifier.

If we consider flow accuracy, we see that for three out of four applications the performance of the two classifiers is comparable. Yet Abacus presents a very low percentage of 13.35% true positives for PPLive, with a rather large number of flows falling in the unknown class. By examining the classification logs, we found that PPLive actually uses more ports on the same host to perform different functions (e.g. one for video transfer, one for overlay maintenance). In particular, from one port it generates many single-packet flows all directed to different peers, apparently to perform peer discovery. All these flows, which account for a negligible portion of the overall bytes, fall in the first bin of the abacus signature, which is always classified as unknown. However, from the byte-wise results we can conclude that the video endpoint is always correctly classified.

Finally, we observe that Kiss has a lower flow accuracy for the operator traces. In fact, the great percentage of flows falling in the "not classified" class means that many flows are shorter than 80 packets. Again, this is only a minor issue since Kiss byte accuracy is anyway very high.

5 Comparison

5.1 Functional Comparison

In the previous section we have shown that the classifiers actually have similar performance for the identification of the target applications as well as the "unknown" traffic. Nevertheless, they are based on very different approaches, both presenting pros and cons, which need to be all carefully taken into account.

Tab. 3 summarizes the main characteristics of the classifiers, which are reviewed in the following. The most important difference is the classification technique used. Even if both classifiers are statistical, they work at different levels and clearly belong to different families of classification algorithms. Abacus is a behavioral classifier since it builds a statistical representation of the pattern of traffic generated by an endpoint, starting from transport-level data. Conversely, Kiss derives a statistical description of

Table 3. Main characteristics of Abacus and Kiss

Characteristic	Abacus	Kiss
Technique	Behavioral	Stocastic Payload Inspection
Entity	Endpoint	Endpoint/Flow
Input Format	Netflow-like	Packet trace
Grain	Fine grained	Fine grained
Protocol Family	P2P-TV	Any
Rejection Criterion	Threshold	Train-based
Train set size	Big *(4000 smp.)*	Small *(300 smp.)*
Time Responsiveness	Deterministic *(5sec)*	Stochastic *(early 80pkts)*
Network Deploy	Edge	Edge/Backbone

the application protocol by inspecting packet-level data, so it is a payload-based classifier.

The first consequence of this different approach lies in type and volume of information needed for the classification. In particular, Abacus takes as input just a measurement of the traffic rate of the flows directed to an endpoint, in terms of both bytes and packets. Not only this represents an extremely small amount of information, but it could also be gathered by a Netflow monitor, so that no packet trace has to be inspected by the classification engine itself. On the other hand, Kiss must necessary access packet payload to compute its features. This constitutes a more expensive operation, even if only the first 12 bytes are sufficient to achieve a high classification accuracy.

Despite the different input data, both classifiers work at a fine-grained level, i.e., they can identify the specific application related to each flow and not just the class of applications (e.g., P2P-TV). This consideration may appear obvious for a payload-based classifier such as Kiss, but it is one of the strength of Abacus over other behavioral classifiers which are usually capable only of a coarse grained classification.

Clearly, Abacus pays the simplicity of its approach in terms of possible target traffic. In fact its classification process relies on some specific properties of P2P-TV traffic (i.e., the steady download rate required by the application to provide a smooth video playback), which are really tied to this particular service. For this reason Abacus currently cannot be applied to applications other than P2P-TV applications. On the contrary, Kiss is more general, it makes no particular assumptions on its target traffic and can be applied to any protocol. Indeed, it successfully classifies other kinds of P2P applications, from file-sharing (e.g., eDonkey) to P2P VoIP (e.g., Skype), as well as traditional client-server applications (e.g., DNS).

Another important distinguishing element is the rejection criterion. Abacus defines an hypersphere for each target class and measures the distance of each classified point from the center of the associated hypersphere by means of the Bhattacharyya formula. Then, by employing a threshold-based rejection criterion, a point is label as “unknown” when its distance from the center exceeds a given value. Instead Kiss exploits a multi-class SVM model where all the classes, included the unknown, are represented in the training set. If this approach makes Kiss very flexible, the characterization of the classes

Table 4. Analytical comparison of the resource requirements of the classifiers

	Abacus	Kiss
Memory allocation	2F *counters*	2^bG *counters*

Packet processing	`EP_state = hash(`IP_d`, `$port_d$`)`
	`FL_state = EP_state.hash(`IP_s`, `$port_s$`)`
	`FL_state.pkts ++`
	`FL_state.bytes += pkt_size`
		`EP_state = hash(`IP_d`, `$port_d$`)`
		for `g = 1 to G` **do**
		`  `P_g` = payload[g]`
		`  EP_state.O[g][`P_g`]++`
		end for

Tot. op.	2 lup + 2 sim	(2G+1) lup + G sim

Feature extraction	`EP_state = hash(`IP_d`, `$port_d$`)`
	for all `FL_state in EP_state.hash` **do**
	`  p[ `log_2`(FL_state.pkts )] += 1`
	`  b[ `log_2`(FL_state.bytes)] += 1`
	end for
	`N = count(keys(EP_state.hash))`
	for all `i = 0 to B` **do**
	`  p[i] /= N`
	`  b[i] /= N`
	end for
		`E = C/`2^b *(precomputed)*
		for `g = 1 to G` **do**
		`  Chi[g] = 0`
		`  `**for** `i = 0 to `2^b **do**
		`  Chi[g] +=`
		`  (EP_state.O[g][i]-E)`^{2}
		`  `**end for**
		`  Chi[g] /= E`
		end for

Tot. op.	(4F+2B+1) lup + 2(F+B) com + 3F sim	2^{b+1}G lup + G com + (3·2^b+1)G sim

lup=lookup, com=complex operation, sim=simple operation

can be critical especially for the unknown since it is important that the training set contains samples from all possible protocols other than the target ones.

We also notice that there is an order of magnitude of difference in the size of the training set used by the classifiers. In fact, we trained Abacus with 4000 samples per class (although in some tests we experimented the same performance even with smaller sets) while Kiss, thanks to the combination of the discriminative power of both the ChiSquare signatures and the SVM decision process, needs only 300 samples per class.

On the other hand, Kiss needs at least 80 packets generated from (or directed to) an endpoint in order to classify it. This may seem a strong constraint but results reported in Sec. 4 actually show that the percentage of not supported traffic is negligible, at least in terms of bytes. This is due to the adoption of the endpoint-to-flow label propagation scheme, i.e. the propagation of the label of an "elephant" flow to all the "mice" flows of the same endpoint. With the exception of particular traffic conditions, this labeling technique can effectively bypass the constraint on the number of packets.

Finally, for what concerns the network deployment, Abacus needs all the traffic received by the endpoint to characterize its behavior. Therefore, it is only effective when placed at the edge of the network, where all traffic directed to an host transits. Conversely, in the network core Abacus would likely see only a portion of this traffic, so gathering an incomplete representation of an endpoint behavior, which in turn could result in an inaccurate classification. Kiss, instead, is more robust with respect to the deployment position. In fact, by inspecting packet payload, it can operate even on a limited portion of the traffic generated by an endpoint, provided that the requirement on the minimum number of packets is satisfied.

Table 5. Numerical case study of the resource requirements of the classifiers

	Abacus	Kiss
Memory allocation	320 bytes	384 bytes
Packet processing	2 lup + 2 sim	49 lup + 24 sim
Feature selection	177 lup + 96 com + 120 sim	768 lup + 24 com + 1176 sim
Params values	*B=8, F=40*	*G=24, b=4*

5.2 Computational Cost

To complete the classifiers comparison, we provide an analysis of the requirements in terms of both memory occupation and computational cost. We follow a theoretical approach and calculate these metrics from the formal algorithm specification. In this way, our evaluation is independent from specific hardware platforms or code optimizations. Tab. 4 compares the costs from an analytical point of view while in Tab. 5 there is a numerical comparison based on a case study.

Memory footprint is mainly related to the data structures used to compute the statistics. Kiss requires a table of $G \cdot 2^b$ counters for each endpoint to collect the observed frequencies employed in the chi-square computation. For the default parameters, i.e. $G = 24$ chunks of $b = 4$ bits, each endpoint requires 384 counters. Abacus, instead, requires two counters for each flow related to an endpoint, so the total amount of memory is not fixed but it depends on the number of flows per endpoint. As an example, Fig. 1-(a) reports, for the two operator traces, the CDF of the number of flows seen by each endpoint in consecutive windows of 5 seconds, the default duration of the Abacus time-window. It can be observed that the 90th percentile in the worst case is nearly 40 flows. By using this value as a worst case estimate of the number of flows for a generic endpoint, we can say that $2 \cdot \#Flows = 80$ counters are required for each endpoint. This value is very small compared to Kiss requirements but for a complete comparison we also need to consider the counters dimension. As Kiss uses windows of 80 packets, its counters assume values in the interval $[0, 80]$ so single byte counters are sufficient. Using the default parameters, this means 384 bytes for each endpoint. Instead, the counters of Abacus do not have a specific interval so, using a worst case scenario of 4 bytes for each counter, we can say that 320 bytes are associated to each endpoint. In conclusion, in the worst case, the two classifiers require a comparable amount of memory but on average Abacus requires less memory than Kiss.

Computational cost can be evaluated comparing three tasks: the operations performed on each packet, the operations needed to compute the signatures and the operations needed to classify them. Tab. 4 reports the pseudo code of the first two tasks for both classifiers, specifying also the total amount of operations needed for each task. The operations are divided in three categories and considered separately as they have different costs: *lup* for memory lookup operations, *com* for complex operations (i.e., floating point operations), *sim* for simple operations (i.e., integer operations).

Let us first focus on the packet processing part, which presents many constraints from a practical point of view, as it should operate at line speed. In this phase, Abacus needs 2 memory lookup operations, to access its internal structures, and 2 integer

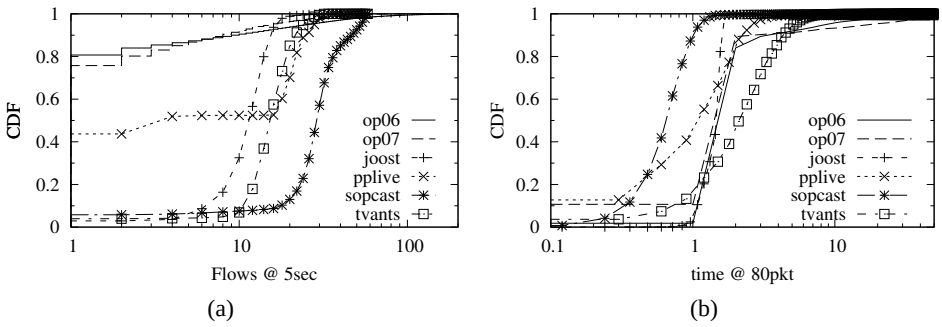

Fig. 1. Cumulative distribution function of (a) number of flows per endpoint and (b) duration of a 80 packet snapshot for the operator traces

increments per packet. Kiss, instead, needs $2G + 1 = 49$ lookup operations, half of which are accesses to packet payload. Then, Kiss must compute G integer increments. Since memory read operations are the most time consuming, from our estimation we can conclude that Abacus should be approximately 20 times faster than Kiss in the packet processing phase.

The evaluation of the signature extraction process instead is more complex. First of all, since the number of flows associated to an endpoint is not fixed, the Abacus cost is not deterministic but, like in the memory occupation case, we can consider 40 flows as a worst case scenario. For the lookup operations, Considering $B = 8$, Abacus requires a total of 177 operations, while Kiss needs 768 operations, i.e., nearly four times as many. For the arithmetic operations, Abacus needs 96 floating point and 120 integer operations, while Kiss needs 24 floating point and 1176 integer operations.

Abacus produces one signature every 5 seconds, while Kiss signatures are processed every 80 packets. To estimate the frequency of the Kiss calculation, in Fig. 1(b) we show the CDF of the amount of time needed to collect 80 packets for an endpoint. It can be observed that, on average, a new signature is computed every 2 seconds. This means that Kiss performs the feature calculation more frequently, i.e., it is more reactive and possibly more accurate than Abacus but obviously also more resource consuming.

Finally, the complexity of the classification task depends on the number of features per signature, since both classifiers are based on a SVM decision process. The Kiss signature is composed, by default, of $G = 24$ features, while the Abacus signature contains 16 features: also from this point of view Abacus appears lighter than Kiss.

6 Conclusions

In this paper we compared two approaches to the classification of P2P-TV traffic. We provided not only a quantitative evaluation of the algorithm performance by testing them on a common set of traces, but also a more insightful discussion of the differences deriving from the two followed paradigms.

The algorithms proved to be comparable in terms of accuracy in classifying P2P-TV applications, at least regarding the percentage of correctly classified bytes. Differences emerged also when we compared the computational cost of the classifiers. With this respect, Abacus outperforms Kiss, because of the simplicity of the features employed to characterize the traffic. Conversely, Kiss is much more general, as it can classify other types of applications as well.

Our work is a first step in cross-evaluating the novel algorithms proposed by the research community in the field of traffic classification. We showed how an innovative behavioral method can be as accurate as a payload-based one, and at the same time lighter, so being a perfect candidate for scenarios with hard constraints in term of computational resources. However, we also showed some limitations in its general applicability, which we would like to address in our future work.

References

1. Moore, Andrew. W. and Papagiannaki, Konstantina: Toward the Accurate Identification of Network Applications. In: Passive and Active Measurement (PAM'05), Boston, MA, US (March 2005)
2. Karagiannis, T., Broido, A., Brownlee, N., Claffy, K., Faloutsos, M.: Is p2p dying or just hiding? In: IEEE GLOBECOM '04., Dallas, Texas, US (2004)
3. Finamore, A., Mellia, M., Meo, M., Rossi, D.: KISS: Stocastic Packet Inspection. In: Traffic Measurement and Analysis (TMA) Workshop at IFIP Networking '09, Aachen, Germany (May 2009)
4. Cascarano, N., Risso, F., Este, A., Gringoli, F., Salgarelli, L., Finamore, A., Mellia, M.: Comparing p2ptv traffic classifiers submitted to IEEE ICC 2010.
5. Valenti, S., Rossi, D., Meo, M., Mellia, M., Bermolen, P.: Accurate, Fine-Grained Classification of P2P-TV Applications by Simply Counting Packets. In: Traffic Measurement and Analysis (TMA) Workshop at IFIP Networking '09, Aachen, Germany (May 2009)
6. Bernaille, L., Teixeira, R., Salamatian, K.: Early application identification. In: Proc. of ACM CoNEXT 2006, Lisboa, PT (December 2006)
7. Williams, N., Zander, S., Armitage, G.: A prelimanery performance comparison of five machine learning algorithms for practical ip traffic flow comparison. ACM SIGCOMM Comp. Comm. Rev. **36**(5) (2006) 7–15
8. Erman, J., Arlitt, M., Mahanti, A.: Traffic classification using clustering algorithms. In: MineNet 06: Mining network data (MineNet) Workshop at ACM SIGCOMM '06, Pisa, Italy (2006)
9. Karagiannis, T., Papagiannaki, K., Faloutsos, M.: Blinc: multilevel traffic classification in the dark. SIGCOMM Comput. Commun. Rev. **35**(4) (2005) 229–240
10. Iliofotou, M., Kim, H., Pappu, P., Faloutsos, M., Mitzenmacher, M., Varghese, G.: Graph-based p2p traffic classification at the internet backbone. In: 12th IEEE Global Internet Symposium (GI2009), Rio de Janeiro, Brazil (April 2009)
11. Salgarelli, L., Gringoli, F., Karagiannis, T.: Comparing traffic classifiers. ACM SIGCOMM Comp. Comm. Rev. **37**(3) (2007) 65–68
12. Nguyen, T.T.T., Armitage, G.: A survey of techniques for internet traffic classification using machine learning. IEEE Communications Surveys & Tutorials **10**(4) (2008) 56–76
13. Kim, H., Claffy, K., Fomenkov, M., Barman, D., Faloutsos, M., Lee, K.: Internet traffic classification demystified: myths, caveats, and the best practices. In: Proc. of ACM CoNEXT 2008, Madrid, Spain (2008)

14. Li, W., Canini, M., Moore, A.W., Bolla, R.: Efficient application identification and the temporal and spatial stability of classification schema. Computer Networks **53**(6) (2009) 790–809
15. Hei, X., Liang, C., Liang, J., Liu, Y., Ross, K.W.: A Measurement Study of a Large-Scale P2P IPTV System. IEEE Transactions on Multimedia (Dec. 2007)
16. Li, B., Qu, Y., Keung, Y., Xie, S., Lin, C., Liu, J., Zhang, X.: Inside the New Coolstreaming: Principles, Measurements and Performance Implications. In: IEEE INFOCOM '08, Phoenix, AZ (April 2008)
17. Cristianini, N., Shawe-Taylor, J.: An introduction to Support Vector Machines and Other Kernel-based Learning Methods. Cambridge University Press, New York, NY (1999)
18. Napa-Wine: `http://www.napa-wine.eu/`.
19. Risso, F., Cascarano, N.: Diffinder available at `http://netgroup.polito.it/research-projects/l7-traffic-classification`.