Reviewing Traffic ClassificationData Traffic Monitoring and Analysis

Reviewing Traffic Classification

Silvio Valenti[1,4], Dario Rossi[1], Alberto Dainotti[2],
Antonio Pescapè[2], Alessandro Finamore[3], Marco Mellia[3]

[1] Telecom ParisTech, France – first.last@enst.fr
[2] Università di Napoli Federico II, Italy – last@unina.it
[3] Politecnico di Torino, Italy – first.last@polito.it
[4] Current affiliation: Google, Inc.

Abstract. Traffic classification has received increasing attention in the last years. It aims at offering the ability to automatically recognize the application that has generated a given stream of packets from the direct and passive observation of the individual packets, or stream of packets, flowing in the network. This ability is instrumental to a number of activities that are of extreme interest to carriers, Internet service providers and network administrators in general. Indeed, traffic classification is the basic block that is required to enable any traffic management operations, from differentiating traffic pricing and treatment (e.g., policing, shaping, etc.), to security operations (e.g., firewalling, filtering, anomaly detection, etc.).

Up to few years ago, almost any Internet application was using well-known transport-layer protocol ports that easily allowed its identification. More recently, the number of applications using random or non-standard ports has dramatically increased (e.g. Skype, BitTorrent, VPNs, etc.). Moreover, often network applications are configured to use well-known protocol ports assigned to other applications (e.g. TCP port 80 originally reserved for Web traffic) attempting to disguise their presence.

For these reasons, and for the importance of correctly classifying traffic flows, novel approaches based respectively on packet inspection, statistical and machine learning techniques, and behavioral methods have been investigated and are becoming standard practice. In this chapter, we discuss the main trend in the field of traffic classification and we describe some of the main proposals of the research community.

We complete this chapter by developing two examples of behavioral classifiers: both use supervised machine learning algorithms for classifications, but each is based on different features to describe the traffic. After presenting them, we compare their performance using a large dataset, showing the benefits and drawback of each approach.

1 Introduction

Traffic classification is the task of associating network traffic with the generating application. Notice that the TCP/IP protocol stack, thanks to a clear repartition between layers, is completely agnostic with respect to the application protocol or to the data carried inside packets. This layered structure has been one of the main reasons for the success of the Internet; nevertheless, sometimes network operators, though logically at

Table 1. Taxonomy of traffic classification techniques

Approach	Properties exploited	Granularity	Timeliness	Comput. Cost
Port-based	Transport-layer port [49, 50, 53]	Fine grained	First Packet	Lightweight
Deep Packet Inspection	Signatures in payload [44, 50, 60]	Fine grained	First payload	Moderate, access to packet payload
Stochastic Packet Inspection	Statistical properties of payload [26, 30, 37]	Fine grained	After a few packets	High, eventual access to payload of many packets
Statistical	Flow-level properties [38, 45, 50, 58]	Coarse grained	After flow termination	Lightweight
	Packet-level properties [8, 15]	Fine grained	After few packets	Lightweight
Behavioral	Host-level properties [35, 36, 67]	Coarse grained	After flow termination	Lightweight
	Endpoint rate [7, 28]	Fine grained	After a few seconds	Lightweight

layer-3, would be happy to know to which application packets belong, in order to better manage their network and to provide additional services to their customers. Traffic classification is also instrumental for all security operations, like filtering unwanted traffic, or triggering alarms in case of an anomaly has been detected.

The information provided by traffic classification is extremely valuable, sometimes fundamental, for quite a few networking operations [38, 42, 46, 52]. For instance, a detailed knowledge of the composition of traffic, as well as the identification of trends in application usage, is required by operators for a better *network design and provisioning. Quality of service* (QoS) solutions [58], which prioritize and treat traffic differently according to different criteria, need first to divide the traffic in different classes: identifying the application to which packets belong is crucial when assigning them to a class. In the same way, traffic classification enables differentiated class *charging* or Service Level Agreements (SLA) verification. Finally, some national governments expect ISPs to perform *Lawful Interception* [6] of illegal or critical traffic, thus requiring them to know exactly the type of content transmitted over their networks. Traffic classification represents in fact the first step for activities such as *anomaly detection* for the identification of malicious use of network resources, and for security operation in general, like firewalling and filtering of unwanted traffic [53, 56].

If, on the one hand, the applications of traffic classification are plentiful, on the other hand, the challenges classifiers have to face are not to be outdone. First, they must deal with an increasing amount of traffic as well as equally increasing transmission rates: to cope with such speed and volume, researchers are looking for *lightweight algorithms* with as little computational requirements as possible. The task is further exacerbated by developers of network applications doing whatever in their power to hide traffic and to elude control by operators: traffic encryption and encapsulation of data in other protocols are just the first two examples that come to mind. Therefore, researchers had to come out with novel and unexpected ways for identifying traffic.

This Chapter is organized as follows. In Section 2, to provide the background of the field, we define a taxonomy in which we highlight the most important contributions in each category along with their most important characteristics. In Section 3 we discuss the state of the art in the field of traffic classification. In Section 4 we describe the most used machine-learning algorithms in the field of traffic classification. In Section 5 we present two antipodean examples of traffic classification approaches, that we directly compare in iSection 6. Section 7 ends the Chapter.

2 Traffic Classification: basic concepts and definitions

The large body of literature about traffic classification [7, 8, 15, 20, 21, 26, 28, 30, 35, 36, 38, 44, 45, 49, 50, 50, 53, 58, 60, 67] is a further evidence of the great interest of the research community towards this topic. In the following, we will present an overview of the different approaches and methodologies that have been proposed by researchers to solve this issue. It is important to underline that this is far from being an attempt to provide a comprehensive list of all papers in this field (which, given their number, would be particularly tedious). Such a detailed reference can be found in a few surveys [38,52] or in related community websites (e.g., [1]). Our aim is rather to identify the most important research directions so far, as well as the most representative milestone works and findings, to better highlight our contribution to this already deeply investigated subject. Still, despite this huge research effort, the community has not put the last word on traffic classification yet, as a number of challenges and questions still remain open.

To better structure this overview, we divide the classifiers in a few categories according to the information on which they base the classification. This widely accepted categorization, which reflects also the chronological evolution followed by research, is summarized in Tab. 1. The table lists the most important works in each category along with their most relevant characteristics. The most important properties of a traffic classifier, which determine its applicability to different network tasks [19], are:

Granularity We distinguish between *coarse-grained* algorithms, which recognize only large family of protocols (e.g. P2P vs non P2P, HTTP vs Streaming) and *fine-grained* classifiers, which, instead, try to identify the specific protocol (e.g. BitTorrent vs eDonkey file-sharing), or even the specific application (e.g. PPlive vs SopCast live streaming).

Timeliness *Early classification* techniques are able to quickly identify the traffic, after a few packets, thus being suited for tasks requiring a prompt reaction (e.g. security). *Late classification* algorithms take longer to collect traffic properties, and in some case they even have to wait for flow termination (i.e., *post mortem* classification): such techniques are indicated for monitoring tasks, such as charging.

Computational cost The processing power needed to inspect traffic and take the classification decision is an important factor when choosing a classification algorithm. In the context of packet processing, the most expensive operation is usually packet memory access, followed by regular expression matching.

3 State of the Art

In the first days of the Internet, identifying the application associated with some network packets was not an issue whatsoever: protocols were assigned to well-known transport-layer ports by IANA [2]. Therefore, **Port-based classification** [49, 50, 53] simply extracted such value from the packet header and then look it up in the table containing the port-application associations. Unfortunately *Port-based* classification has become largely unreliable [34, 50]. In fact, in order to circumvent control by ISPs, modern applications, especially P2P ones, either use non-standard ports, or pick a random port at startup. Even worse, they hide themselves behind ports of other protocols – this might enable bypassing firewalls as well. While port-based classification may still be reliable for some portion of the traffic [38], nevertheless it will raise undetectable false-positive (e.g., a non-legitimate application hiding beyond well-known port numbers) and false-negative (e.g., a legitimate application running on non-standard ports) classifications.

To overcome this problem, **Payload-based classifiers** [26, 30, 44, 50, 60] were proposed. They inspect the content of packets well beyond the transport layer headers, looking for distinctive hints of an application protocol in packet payloads. We actually split this family of classification algorithms in two subcategories, *Deep packet inspection* (DPI) techniques that try to match a deterministic set of signatures or regular expressions against packet payload, and *Stochastic packet inspection* (SPI), rather looking at the statistical properties of packet content.

DPI has long provided extremely accurate results [50] and has been implemented in several commercial software products as well as in open source projects [4] and in the Linux kernel firewall implementation [3]. The payload of packets is searched for known patterns, keywords or regular expressions which are characteristic of a given protocol: the website of [3] contains a comprehensive lists of well known patterns. Additionally, DPI is often used in intrusion detection systems [53] as a preliminary step to the identification of network anomalies. Besides being extremely accurate, DPI has been proved to be effective from the very first payload packets of a session [5, 54], thus being particularly convenient for early classification.

Despite its numerous advantages, DPI has some significant drawbacks. First the computational cost is generally high, as several accesses to packet memory are needed and memory speed is long known to represent the bottleneck of modern architectures [66]. String and regular expression matching represent an additional cost as well: although there exist several efficient algorithms and data structures for both string matching and regular expression, hardware implementation (e.g. FPGA), ad hoc coprocessors (e.g. DFA) possibly massively parallel (e.g., GPU) are often required to keep up with current transmission speed [41]. These hardware-based approaches have been analyzed and used to improve the performance of machine learning algorithms, traffic classification approaches, and platforms for network security [11,32,43,62,64,68] Yet, it is worth noting that while [64] estimate that the amount of GPUs power can process up to 40 Gbps worth of traffic, bottlenecks in the communication subsystem between the main CPU and the GPU crushes the actual performance down to a mere 5.2 Gbps [64]. Similarly, Network Processors [43] and [62] achieve 3.5 Gbps and 6 Gbps of aggregated traffic rate at most. As we will see, statistical classification outperforms these classification rates without requiring special hardware. Another drawback of DPI is that keywords or

patterns usually need to be derived manually by visual inspection of packets, implying a very cumbersome and error prone trial and error process. Last but not least, DPI fails by design in the case of encrypted or obfuscated traffic.

Stochastic packet inspection (SPI) tries to solve some of these issues, for instance by providing methods to automatically compute distinctive patterns for a given protocol. As an example, authors of [44] define Common Substring Graphs (CSG): an efficient data structure to identify a common string pattern in packets. Other works instead directly apply statistical tools to packet payload: authors of [30] directly use the values of the first payload bytes as features for machine learning algorithms; in [26], instead, a Pearson Chi-square test is used to study the randomness of the first payload bytes, to build a model of the syntax of the protocol spoken by the application. Additionally, this last algorithm is able to deal with protocols with partially encrypted payload, such as Skype or P2P-TV applications.

Authors of [37], instead, propose a fast algorithm to calculate the entropy of the first payload bytes, by means of which they are able to identify the type of content: low, medium and high values of the entropy respectively correspond to text, binary and encrypted content. Authors argue that, even if this is a very rough repartition of traffic and moreover some applications are very likely to use all of these kinds of content, nonetheless such information might reveal useful to prioritize some content over the others (e.g. in enterprise environments, binary transfers corresponding to application updates to fix bugs deserve an high priority). Yet, SPI is still greedy in terms of computational resources, requiring several accesses to packet payload, though with simpler operations (i.e., no pattern matching).

While both [26, 37] use entropy-based classification, a notable difference is represented by the fact that in [26] entropy is computed for chunks of data *across* a stream of packets, while [37] computes entropy over chunks *whitin* the same packet.

Statistical classification [8, 9, 15, 17, 18, 45, 48, 58, 65] is based on the rationale that, being the nature of the services extremely diverse (e.g., Web vs VoIP), so will be the corresponding traffic (e.g., short packets bursts of full-data packets vs long, steady throughput flows composed of small-packets). Such classifiers exploit several flow-level measurements, a.k.a. *features*, to characterize the traffic of the different applications [45,48,58]: a comprehensive list of a large number of possible traffic discriminators can be found in the technical report [47]. Finally, to perform the actual classification, statistical classifiers apply data mining techniques to these measurements, in particular machine learning algorithms.

Unlike payload-based techniques, these algorithms are usually very lightweight, as they do not access packet payload and can also leverage information from flow-level monitors such as [12]. Another important advantage is that they can be applied to encrypted traffic, as they simply do not care what the content of packets is. Nevertheless, these benefits are counterbalanced by a decrease in accuracy with respect to DPI techniques, which is why statistical-based algorithms have not evolved to commercial products yet. Still, researchers claim that in the near future operators will be willing to pay the cost of a few errors for a much lighter classification process.

We can further divide this class of algorithms in a few subclasses according to the data mining techniques employed and to the protocol layer of the features used. Con-

cerning the first criterion, on one hand, unsupervised clustering of traffic flows [45] (e.g., by means of the K-means algorithm) does not require training and allows to group flows with similar features together, possibly identifying novel unexpected behaviors; on the other hand, supervised machine learning techniques [38,65] (e.g., based on Naive Bayes, C4.5 or Support Vector Machines) need to be trained with already classified flows, but are able to provide a precise labeling of traffic. Regarding the protocol layer, we have classifiers employing only flow-level features [48] (e.g., duration, total number of bytes transferred, average packet-size), as opposed to algorithms using packet-level features [8, 15] (e.g., size and direction of the very first packets of a flow). The former ones are usually capable of late (in some cases only *post-mortem*), coarse-grained classification, whereas the latter ones can achieve early, fine-grained classification.

Finally, **Behavioral classification** [35, 36, 67] moves the point of observation further up in the network stack, and looks at the whole traffic received by a host, or an (IP:port) endpoint, in the network. By the sole examination of the generated traffic patterns (e.g., how many hosts are contacted, with which transport layer protocol, on how many different ports) behavioral classifiers try to identify the application running on the target host. The idea is that different applications generate different patterns: for instance, a P2P host will contact many different peers typically using a single port for each host, whereas a Web server will be contacted by different clients with multiple parallel connections.

Some works [35, 67] characterize the pattern of traffic at different levels of detail (e.g., social, functional and application) and employ heuristics (such as the number of distinct ports contacted, or transport-layer protocols used) to recognize the class of the application running on a host (e.g., P2P vs HTTP). Works taking the behavioral approach to its extreme analyze the graph of connections between endpoints [31, 33], showing that P2P and client-server application generate extremely different connection patterns and graphs. They prove also that such information can be leveraged to classify the traffic of these classes of services even in the network core. A second group of studies [7, 28], instead, propose some clever metrics tailored for a specific target traffic, with the purpose of capturing the most relevant properties of network applications. Combining these metrics with the discriminative power of machine learning algorithms yields extremely promising results. The Abacus classifier [7] belongs to this last family of algorithms, and it is the first algorithm able to provide a fine-grained classification of P2P applications.

Behavioral classifiers have the same advantages of statistical-based classifiers, being lightweight and avoiding access to packet payload, but are usually able to achieve the same accuracy with even less information. Such properties make them the perfect candidate for the most constrained settings. Moreover given the current tendency toward flow-level monitors such as NetFlow [12], the possibility to operate on the sole basis of behavioral characteristics is a very desirable property for classifiers.

We wrap up this overview with an overall consideration on the applicability of classifiers. With few exceptions such as [24], the wide majority of the classification algorithms proposed in literature cannot be directly applied to the network core. Limitations can be either intrinsic to the *methodology* (e.g., behavioral classification typically focuses on endpoint [67] or end-hosts [36] activity), or be tied to the *computational com-*

plexity (e.g., DPI [26, 44, 50, 60] cannot cope with the tremendous amount of traffic in the network core), or to *state scalability* (e.g., flow-based classification [45,48] requires to keep a prohibitive amount of per-flow state in the core), or to *path changes* (path instabilities or load balancing techniques can make early classifications techniques such as [8, 15] fail in the core). At the same time, we point out that classifying traffic at the network ingress point is a reasonable choice for ISPs: indeed, traffic can be classified and tagged at the access (e.g., DiffServ IP TOS field, MPLS, etc.), on which basis a differential treatment can then be applied by a simple, stateless and scalable core (e.g., according to the class of application.). We investigate deeper this issue in the second part of this dissertation.

Finally we must deal with a transversal aspect of traffic classification. The heterogeneity of approaches, the lack of a common dataset and of a widely approved methodology, all contribute to make the comparison of classification algorithms a daunting task [59]. In fact, to date, most of the comparison effort has addressed the investigation of different machine learning techniques [8, 23, 65], using the same set of features and the same set of traces. Only recently, a few works have specifically taken into account the comparison problem [10, 38, 42, 52]. The authors of [52] present a qualitative overview of several machine learning based classification algorithms. On the other hand, in [38] the authors compare three different approaches (i.e., based on signatures, 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 [42], where authors evaluate spatial and temporal portability of a port-based, a DPI and a flow-based classifier.

4 Machine-Learning Algorithms for Traffic Classification

In this section we will briefly introduce the problem of traffic classification in machine learning theory (with a particular focus on the algorithms we actually employed to exemplify the traffic classification performance in 6), all falling in the category of *supervised classification*.

There is a whole field of research on machine learning theory which is dedicated to supervised classification [40], hence it is not possible to include a complete reference in this chapter. Moreover, instead of improving the classification algorithms themselves, we rather aim at taking advantage of our knowledge of network applications to identify good properties, or features, for their characterization. However, some basic concepts are required to correctly understand how we applied machine learning to traffic classification.

A supervised classification algorithm produces a function f, *the classifier*, able to associate some input data, usually a vector $\mathbf{x}$ of numerical attributes x_i called *features*, to an output value c, the class label, taken from a list C of possible ones. To build such a mapping function, which can be arbitrary complex, the machine learning algorithm needs some examples of already labeled data, the *training set*, i.e. a set of couples $(\mathbf{x}, c)$ from which it *learns* how to classify new data. In our case the features x_i are distinctive properties of the traffic we want to classify, while the class label c is the application associated with such traffic.

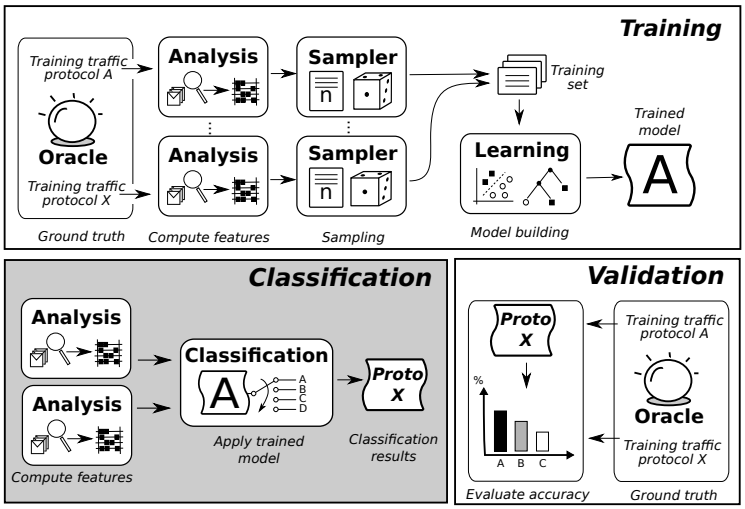


Fig. 1. Common workflow of supervised classification.

From a high-level perspective, supervised classification consists of three consecutive phases which are depicted in Fig. 1. During the *training phase* the algorithm is fed with the training set which contains our reference data, the already classified training points. The selection of the training points is a fundamental one, with an important impact on the classifier performance. Extra care must be taken to select enough representative points to allow the classifier to build a meaningful model; however, including too many points is known to degenerate in *overfitting*, where a model is too finely tuned and becomes "picky", unable to recognize samples which are just slightly different from the training ones.

Notice that, preliminary to the training phase, an *oracle* is used to associate the protocol label with the traffic signatures. Oracle labels are considered accurate, thus representing the *ground truth* of the classification. Finding a reliable ground truth for traffic classification is a research topic on its own, with not trivial technical and privacy issues and was investigated by a few works [16, 29].

The second step is the *classification phase*, where we apply the classifier to some new samples, the *test set*, which must be disjoint from the training set. Finally a third phase is needed to *validate* the results, comparing the classifiers outcome against the reference ground truth. This last phase allows to assess the expected performance when deploying the classifier in operational networks.

In this chapter we describe two of the supervised classification algorithms most used in traffic classification literature, namely *Support Vector Machines* and *Classification trees*. This choice is not only based on their large use in the literature of traffic classification, but as they are recognized as having the largest discriminative power in the machine learning community. Specifically, classification accuracy of Support Vector Machines and Classification trees has been compared in [38, 61]: Support Vector Machines exhibit the best classification performance in [38], while in [61] the authors

show the superior performance of *Classification trees*. As for the complexity of these approaches,

As for the complexity of these techniques, authors in [22] show how statistical classification based on *Classification trees* can sustain a throughput in excess of 10 Gbps on off-the-shelf hardware, thus outperforming the current state of the art employing GPUs for DPI classification [43, 62, 64]. The next subsections further elaborate the computational complexity of each technique.

4.1 Support Vector Machine

Support Vector Machine (SVM), first proposed by Vapnik [13], is a binary supervised classification algorithm which transforms a non-linear classification problem in a linear one, by means of what is called a "kernel trick". In the following we intuitively explain how SVM works and refer the reader to [14, 65] for a more formal and complete description of the algorithm.

SVM interprets the training samples as points in a multi-dimensional vector space, whose coordinates are the components of the feature vector $\mathbf{x}$. Ideally we would like to find a set of surfaces, partitioning this space and perfectly separating points belonging to different classes. However, especially if the problem is non-linear, points might be spread out in the space thus describing extremely complex surface difficult, when not impossible, to find in a reasonable time. The key idea of SVM is then to map, by means of a kernel function, the training points in a newly transformed space, usually with higher or even infinite dimensionality, where points can be separated by the easiest surface possible, an hyperplane. In the target space, SVM must basically solve the optimization problem of finding the hyperplane which (i) separates points belonging to different classes and (ii) has the maximum distance from points of either class. The training samples that fall on the margin and identify the hyperplane are called *Support Vectors* (SV).

At the end of the training phase SVM produces a model, which is made up of the parameters of the kernel function and of a collection of the support vectors describing the partitioning of the target space. During the classification phase, SVM simply classifies new points according to the portion of space they fall into, hence classification is much less computationally expensive than training. Since natively SVM is a binary classifier, some workaround is needed to cope with multi-class classification problems. The strategy often adopted is the *one-versus-one*, where a model for each pair of classes is built and the classification decision is based on a majority voting of all binary models.

Support Vector Machines have proved to be an effective algorithm yielding good performance out-of-the-box without much tuning, especially in complex feature spaces, and has showed particularly good performance in the field of traffic classification [38, 65]. Several kernel functions are available in literature but usually Gaussian kernel exhibits the best accuracy. One drawback of SVM is that models in the multidimensional space cannot be interpreted by human beings and it is not possible to really understand the reason why a model is good or bad. Another, more important, drawback is that the classification process may still require a fair amount of computation. Specifically, the number of operations to be performed is linear in the number of SVs (i.e., the represen-

tative samples) per each class. When the number of classes is large (say, in the order of 100s or 1000s applications), the computational cost can be prohibitive

4.2 Decision Trees

Decision Trees [39] represent a completely orthogonal approach to the classification problem, using a tree structure to map the observation input to a classification outcome. Again, being this a supervised classification algorithms, we have the same three phases: training, testing and validation.

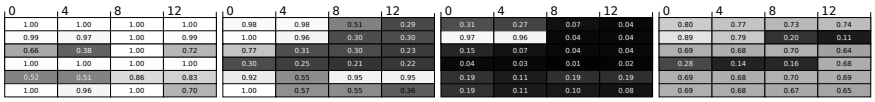
During the training phase the algorithm builds the tree structure from the sample points: each intermediate node (a.k.a. split node) represents a branch based on the value of one feature, while each leaf represents a classification outcome. The classification process, instead, consists basically in traversing the tree from the root to the leaves with a new sample, choosing the path at each intermediate node according to the criteria individuated by the training phase. Like in SVM, the classification process is way more lightweight than the learning phase. One big advantage of this algorithm over SVM is that the tree can be easily read and eventually interpreted to understand how the algorithms leverages the features for the classification. Another advantage is that classification is based on conditional tests and if-then-else branches, which make it computationally very efficient with respect to SVM.

Literature on this subject contains quite a few decision tree building algorithms, which differ in the way they identify the feature and threshold value for the intermediate split nodes. The best known example of classification tree is the C4.5 algorithm [39], which bases such selection on the notion of *Information Gain*. This is a metric from information theory which measures how much information about the application label is carried by each features, or, in other words, how much the knowledge of a feature tells you about the value of the label. We delay a formal definition of the information gain metric to the next chapter, where we take advantage of it for feature selection purposes. After calculating the information gain of each feature for the training set points, C4.5 picks as splitting feature for each node the one which maximizes such a score: this strategy of using the most helpful attributes at each step is particular efficient, yielding trees of very limited depth (since the most critical split nodes are located toward the top of the tree), which further simplify the computational requirement.

5 Two antipodean examples.

In this section, we overview a couple of techniques we propose for the online classification of traffic generated by P2P applications (and, possibly, non-P2P application as well).

We mainly consider two approaches with radically different designs. One approach, named Kiss [25, 26], is *payload* based: it inspects the packet payload to automatically gather a stochastic description of the content, thus inferring the *syntax* of the application protocol rather than payload *semantic*. The other approach, named Abacus [7, 63], is instead *behavioral*: it analyzes the transport level exchanges of P2P applications, discriminating between different *protocol dynamics*.



(a) Joost (b) SopCast (c) TVAnts (d) PPLive

Fig. 2. Mean kiss signatures, 24 chunks of 4 bits each (higher value and lighter color correspond to higher determinism)

Both Kiss and Abacus achieve very reliable classification but, in reason of their different design, have their pros and cons. For instance, payload-based classification fails when data is fully encrypted (e.g., IPsec, or encrypted TCP exchanges), while the behavioral classifier is unable to classify a single flow (i.e., as protocol dynamics need the observation of multiple flows). A detailed comparison of both techniques is reported in Sec. 6

5.1 Kiss: Stochastic payload-based classification

High-level idea The first approach we consider is based on the analysis of packet payload, trying to detect the syntax of the application protocol, rather that its semantic. The process is better understood by contrasting it with DPI, which typically searches keywords to identify a specific protocol. With a human analogy, this corresponds to trying to recognize the foreign language of an overheard conversation by searching for known words from a small dictionary (e.g., “Thanks” for English language, “Merci” for French, “Grazie” for Italian and so on).

The intuition behind Kiss is that application-layer protocols can however be identified by statistically characterizing the stream of bytes observed in a flow of packets. Kiss automatically builds protocol signatures by measuring entropy (or Chi-Square test) of the packet payload. Considering the previous analogy, this process is like recognizing the foreign language by considering only the cacophony of the conversation, letting the protocol syntax emerge, while discarding its actual semantic.

Fig. 2 reports examples of mean Kiss signatures for popular P2P-TV applications like PPLive, SopCast, TVAnts and Joost that we will use often as examples in this Chapter (and for the comparison in Sec. 6). The picture represents the application layer header, where each group of 4 bits is individually considered: for each group, the amount of entropy is quantified by means of a Chi-Square test χ^2 with respect to the uniform distribution. The syntax of the header is easy to interpret: low χ^2 scores hint to high randomness of the corresponding group of bit, due to obfuscation or encryption; high χ^2 scores instead are characteristic of deterministic fields, such as addresses or identifiers; intermediate values correspond to changing fields, such as counters and flags, or groups of bits that are split across field boundaries. As protocol languages are different, Kiss signatures allow to easily distinguish between applications as emerges from Fig. 2.

Formal signature definition Syntax description is achieved by using a simple Chi-Square like test. The test originally estimates the goodness-of-fit between observed samples of a random variable and a given theoretical distribution. Assume that the possible outcomes of an experiment are K different values. Let O_k be the empirical frequencies of the observed values, out of C total observations ($\sum_k O_k = C$). Let E_k be the number of expected observations of k for the theoretical distribution $E_k = C \cdot p_k$ with p_k the probability of value k. Given that C is large, the distribution of the random variable:

$$X = \sum_{k=1}^{K} \frac{(O_k - E_k)^2}{E_k} \tag{1}$$

that represents the distance between the observed empirical and theoretical distributions, can be approximated by a Chi-Square, or χ^2, distribution with $K - 1$ degrees of freedom. In the classical goodness of fit test, the values of X are compared with the typical values of a Chi-Square distributed random variable: the frequent occurrence of low probability values is interpreted as an indication of a bad fitting. In Kiss, we build a similar experiment analyzing the content of groups of bits taken from the packet payload we want to classify.

Chi-Square signatures are built from *streams* of packets. The first N bytes of each packet payload are divided into G *groups* of b consecutive bits each; a group g can take integer values in $[0, 2^b - 1]$. From packets of the same stream, we collect, for each group g, the number of observations of each value $i \in [0, 2^b - 1]$; denote it by $O_i^{(\mathrm{g})}$. We then define a window of C packets, in which we compute:

$$X_g = \sum_{i=0}^{2^b-1} \frac{\left(O_i^{(\mathrm{g})} - E_i^{(g)}\right)^2}{E_i^{(g)}} \tag{2}$$

and collect them in the Kiss signature vector (where, by default, $N = 12, G = 24, b = 4, C = 80$):

$$\overline{X} = [\, X_1,\, X_2,\, \cdots,\, X_G \,] \tag{3}$$

Once the signatures are computed, one possibility to characterize a given protocol is to estimate the expected distribution $\{E_i^{(g)}\}$ for each group g, so that the set of signatures are created by describing the expected distribution of the protocols of interest in the database. During the classification process then, the observed group g distribution $\{O_i^{(g)}\}$ must be compared to each of the $\{E_i^{(g)}\}$ in the database, for example using the Chi-square test to select the most likely distribution. However, this process ends up in being very complex, since (2) must be computed for each protocol of interest.

In addition to the high complexity, the comparison with reference distributions fails when the application protocol includes constant values which are randomly extracted for each flow. For example, consider a randomly extracted "flow ID" in a group. Consider two flows, one used for training and one for testing, generated by the same application. Let the training flow packets take the value 12 in that group. Let the test flow packets take instead the value 1 in the same group. Clearly, the comparison of the two observed distributions does not pass the Chi-square test, and the test flow is not correctly classified as using the same protocol as the training flow.

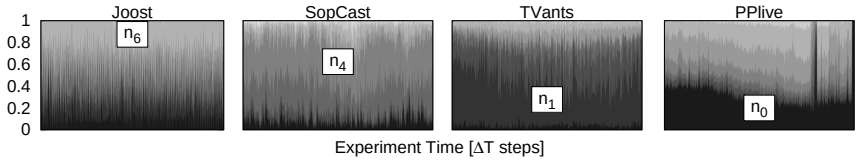


Fig. 3. Temporal evolution of Abacus signatures. Darker color correspond to low order bins, carrying less traffic. Bins are exponential so that $X_i \propto 2^i$, and a mark denotes the most likely bin.

For the above reasons, we propose to simply compare the distance between the observed values and a reference distribution, which we choose as the uniform distribution, i.e., $E_i^{(g)} = E = \frac{C}{2^b}$. In the previous example, the group randomness of the two flows have the same X value, that identify a "constant" field, independently of the actual value of that group. In other terms, we use a Chi-Square like test to measure the randomness of groups of bits, as an implicit estimate of the source entropy.

5.2 Abacus: Fine-grained behavioral classification

High-level idea The Abacus classifier leverages instead on the observation that applications perform different concurrent activities at the same time. Considering for the sake of the example P2P applications, one activity, namely *signaling*, is needed for the maintenance of the P2P infrastructure and is common to all applications. Still, P2P applications differ in the way they actually perform the signaling task, as this is affected by the overlay topology and design (e.g., DHT lookup versus an unstructured flooding search) and by implementation details (e.g., packet size, timers, number of concurrent threads.)

The *data-exchange* activity in instead related to the type of offered service (e.g., file sharing, content, VoIP, VoD, live streaming, etc.). Again, applications are remarkably different, both considering implementation details (e.g., codec, transport layer, neighborhood size, etc.) or the offered service (e.g., low and relatively stable throughput for P2P-VoIP, higher but still relatively stable aggregated incoming throughput for P2P-VoD and TV, largely variable throughput for file-sharing, etc).

Such difference are so striking, that it is actually possible to finely differentiate between different P2P applications offering the same service: in what follows, we make an explanatory example on P2P-TV applications. We again consider P2P-TV applications and contrast the possible ways in which they implement the live TV service. Concerning video transfers, for example, some application may prefer to download most of the video content from a few peers, establishing long-lived flows with them, whereas other applications may prefer to download short fixed-sized "chunks" of video from many peers at the same time. Similarly, some application may implement a very aggressive network probing and discovering policy, constantly sending small-size messages to many different peers, while others may simply contact a few super-peers from which they receive information about the P2P overlay. Continuing our human analogy,

we may say that some peers will be "shy" and contact a few peers, possibly downloading most of the data from them, while others will be "easy-going" and contact many peers, possibly downloading a few data from each.

These differences are shown in Fig. 3, which depicts the temporal evolution of (a simplified version of) the signature used for traffic classification. To capture the above differences, we asses the shyness of a peer P by gauging the proportion of peers that send to P a given amount of traffic in the range $X_i = [X_i^-, X_i^+]$. We then evaluate an empirical probability mass function p_i (pmf) by normalizing the count n_i of peers sending $x \in X_i$ traffic (e.g., packets or bytes), and by ordering the bins such that $X_{i-i}^+ \leq X_i^-$, i.e. low order bins contain less traffic.

In Fig. 3, darker colors correspond to lower bins, and bins are staggered so that they extend to 1 (due to pmf): for the sake of readability, the most likely (i.e., $\text{argmax}_i n_i$) bin is indicated with a textbox. From Fig. 3, it can be seen that each application has a behavior that, although not stationary over time, is however remarkably different from all the others.

Formal signature definition In the following, we restrict our attention to UDP traffic, although endpoint identification can be extended to applications relying on TCP at the transport layer as well[5]. Let us consider the traffic received by an arbitrary end-point $p = (IP, port)$ during an interval of duration ΔT. We evaluate the amount of information received by p simply as the number of received *packets* (although the concept can be extended to the amount of *bytes*, to build more precise signatures [57]).

We partition the space $\mathbb{N}$ of the number of packets sent to p by another peer into $B_n + 1$ bins of exponential-size with base 2: $I_0 = (0, 1]$, $I_i = (2^{i-1}, 2^i]$ for $i = 1, \ldots, B_{n-1}$ and $I_{B_n} = (2^{B_n - 1}, \infty]$. For each ΔT interval, we count the number N_i of peers that sent to p a number of packets $n \in I_i$; i.e., N_0 counts the number of peers that sent exactly 1 packet to p during ΔT; N_1 the number of peers that sent 2 packets; N_2 the number of peers that sent 3 or 4 packets and, finally, N_{B_n} the number of peers that sent at least $2^{B_n - 1} + 1$ packets to p. Let K denote the total number of peers that contacted p in the interval. The behavioral signature is then defined as $\underline{n} = (n_0, \ldots, n_{B_n}) \in \mathbb{R}^{B_n + 1}$, where:

$$n_i = \frac{N_i}{\sum_{j=0}^{B_n} N_j} = \frac{N_i}{K} \tag{4}$$

Since $\underline{n}$ has been derived from the pure count of exchanged packets, we name it "Abacus", which is also a shorthand for "Automated Behavioral Application Classification Using Signatures". Formally, the signature $\underline{n}$ is the observed probability mass function (pmf) of the number of peers that sent a given number of packets to p in a time interval of duration ΔT (where by default $\Delta T = 5, B = 8$).

[5] In case TCP is used, the client TCP port is ephemeral, i.e., randomly selected by the Operating System for each TCP connection. The TCP case would require more complex algorithms in case of traffic *generated* from a specific peer, since ephemeral ports differ among flows generated by the same peer. However, the problem vanishes by focusing on the downlink direction: in this case, we aggregate all traffic *received* by a TCP server port, that is the same for all flows of any given peer.

Table 2. 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

This function is discretized according to the exponential bins described above. The choice of exponential width bins reduces the size of the signature, while keeping the most significant information that can be provided by the pmf. In fact, as the binning is much finer for short number of packets, short flows with even a small difference in the number of packets are likely to end up (e.g. flows composed by a single packet, two packets and three packets are counted respectively in the component n_0, n_1 and n_2). On the contrary, longer flows are coarsely grouped together in the higher bins. Intuitively it is more valuable to distinguish between short flows (e.g., distinguishing between single-packet probes versus short signaling exchanges spanning several packets), while there is no gain in having an extreme accuracy when considering long flows (e.g., distinguishing between 500 or 501 packet long flows). This intuition is discussed in [7], where we examine the impact of different binning strategies.

6 Kiss vs Abacus

At last, we perform a comparison of both approaches, at several levels. To dress a 2π radians view[6], we consider not only the (i) classification results, but also (ii) functional as well as (iii) complexity aspects. To perform the comparison of the classification results, we consider a common subset of traffic, namely that usual set of P2P-TV applications.

In brief, the algorithms are comparable in terms of accuracy in classifying P2P-TV applications, at least regarding the percentage of correctly classified bytes. Differences instead emerged 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.

6.1 Methodology

We evaluate the two classifiers on the traffic generated by the common set of P2P-TV applications, namely PPLive, TVAnts, SopCast and Joost. 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 [51]. For each application we conduct an hour-long experiment where several machines provided by the project partners run the

[6] Well, I assume that since "360° degree" is a common saying for everybody, " 2π radians" should not be an uncommon saying among scientists and engineers.

Table 3. Classification results: Bytewise confusion matrix for Abacus (left) and Kiss (right)

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

=PPLive, =Tvants, =Sopcast, =Joost, un=Unknown, nc=not-classified

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 the traces contain P2P-TV traffic only. This set, available to the research community in [51] 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. 2 the main characteristics of the traces.

To compare the classification results, we employ the `diffinder` tool [55], as already done in [10]. 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 enabling further analysis.

6.2 Classification results

Tab. 3 reports the accuracy achieved by the two classifiers on the test traces using Support Vector Machines (SVM) [14] as learning technique. 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

Table 4. Functional comparison of Abacus and Kiss

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

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 $C = 80$ packets for any endpoint).

Is easy to grasp that both the classifiers are extremely accurate, as most of the bytes are correctly classified (flow accuracy is analyzed in [27]). 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.

6.3 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 and that are summarized in Tab. 4.

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 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 necessarily access packet payload for feature computation: 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. 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, as its classification process relies on some specific properties of P2P traffic. 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 not only other P2P applications (e.g., eDonkey Skype, etc.), but traditional client-server applications (e.g., DNS, RTP, etc.) as well.

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 distance. 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 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 classification performance even with smaller training sets) while Kiss needs only about 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 [26] actually shows that the percentage of not supported traffic is negligible, at least in terms of bytes.

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.

6.4 Computational cost

To complete the classifiers comparison, we provide an analysis of the requirements in terms of both memory occupation and computational cost. We calculate these metrics from the formal algorithm specification, so that our evaluation is independent from specific hardware platforms or code optimizations. Tab. 5 compares the costs in a general case, reporting in the bottom portion specific figures for the default parameters.

Table 5. Computational complexity and resource requirements comparison

	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
Memory allocation	320 bytes	384 bytes
Packet processing	2 lup + 2 sim	49 lup + 24 sim
Feature extraction	177 lup + 96 com + 120 sim	768 lup + 24 com + 1176 sim
	Default params: *B=8, F=40*	Default params: *G=24, b=4*

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

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. 4-(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

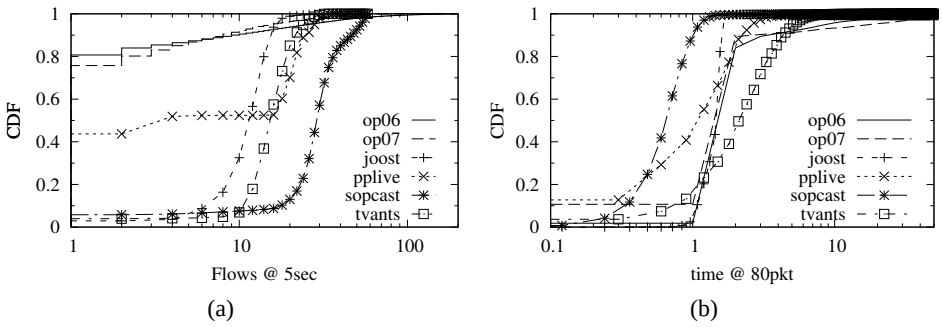


Fig. 4. Cumulative distribution function of (a) number of flows per endpoint and (b) duration of a 80 packet snapshot for the operator traces

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 though 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. 5 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 phase, 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 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, we can conclude that Abacus should be approximately 20 times faster than Kiss in this 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 signatures every $\Delta T = 5$ seconds, while Kiss signatures are processed every $C = 80$ packets. To estimate the frequency of the Kiss calculation, in Fig. 4(b) we show the CDF of the amount of time needed to collect 80 packets for an endpoint: on average, a new signature is computed every 2 seconds. This means that Kiss calculate feature more frequently than Abacus: i.e., it is more reactive 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.

6.5 Summary of comparison

We have described, analyzed and compared Kiss and Abacus, two different approaches for 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 prove to be comparable in terms of accuracy in classifying P2P-TV applications, at least regarding the percentage of correctly classified bytes. Differences emerge 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.

7 Conclusion

In this Chapter we have reviewed literature in the field of traffic classification, a topic which has increased a lot in relevance during last years. Traffic classification is the building block to enable visibility into the traffic carried by the network, and this it is the key element to empower and implement any traffic management mechanisms: service differentiation, network design and engineering, security, accounting, etc., are all based on the assumption to be able to classify traffic.

Research on Internet traffic classification has produced creative and novel approaches. Yet, as described in this Chapter, there is still room for improvements and contributions in the light of classification techniques and platforms, ground truth, comparison approaches, etc. In particular, the natural evolution of the Internet in which novel applications, protocols and habits are born, proliferate and die, calls for a continuous need to update traffic classification methodologies. This is particular critical considering security aspects in which every bit, byte and packet must be checked.

References

1. CAIDA, The Cooperative Association for Internet Data Analysis. `http://www.caida.org/research/traffic-analysis/classification-overview/`.

2. IANA, List of assigned port numbers. http://www.iana.org/assignments/port-numbers,.
3. l7filter, Application layer packet classifier for Linux. http://l7-filter.clearfoundation.com/,.
4. Tstat, http://tstat.tlc.polito.it.
5. G. Aceto, A. Dainotti, W. d. Donato, and A. Pescapè. Portload: Taking the best of two worlds in traffic classification. In *INFOCOM IEEE Conference on Computer Communications Workshops, 2010*, pages 1–5, 15 2010.
6. F. Bakerand, B. Fosterand, and C. Sharp. Cisco Architecture for Lawful Intercept in IP Networks. IETF RFC 3924(Informational), Oct 2004.
7. Paola Bermolen, Marco Mellia, Michela Meo, Dario Rossi, and Silvio Valenti. Abacus: Accurate behavioral classification of P2P-TV traffic. *Elsevier Computer Networks*, 55(6):1394 – 1411, 2011.
8. Laurent Bernaille, Renata Teixeira, and Kave Salamatian. Early application identification. In *Proc. of ACM CoNEXT 2006*, Lisboa, PT, December 2006.
9. V. Carela-Espaoll, P. Barlet-Ros, M. Sole-Simo, A. Dainotti, W. de Donato, and A. Pescapè. K-dimensional trees for continuous traffic classification. pages 141–154, 2010.
10. N. Cascarano, F. Risso, A. Este, F. Gringoli, L. Salgarelli, A. Finamore, and M. Mellia. Comparing P2PTV Traffic Classifiers. In *Communications (ICC), 2010 IEEE International Conference on*, pages 1 –6, may 2010.
11. Niccolo Cascarano, Pierluigi Rolando, Fulvio Risso, and Riccardo Sisto. infant: Nfa pattern matching on gpgpu devices. *Computer Communication Review*, 40(5):20–26, 2010.
12. B. Claise. Cisco Systems NetFlow Services Export Version 9. RFC 3954 (Informational), Oct 2004.
13. Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20:273–297, 1995.
14. Nello Cristianini and John Shawe-Taylor. *An introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, New York, NY, 1999.
15. Manuel Crotti, Maurizio Dusi, Francesco Gringoli, and Luca Salgarelli. Traffic classification through simple statistical fingerprinting. *ACM SIGCOMM Computer Communication Review*, 37(1):5–16, January 2007.
16. Alberto Dainotti, Walter de Donato, and Antonio Pescapé. Tie: A community-oriented traffic classification platform. In *Traffic Monitoring and Analysis*, volume 5537 of *Lecture Notes in Computer Science*, pages 64–74. 2009.
17. Alberto Dainotti, Walter de Donato, Antonio Pescapè, and Pierluigi Salvo Rossi. Classification of network traffic via packet-level hidden markov models. pages 1–5, 30 2008 2008.
18. Alberto Dainotti, Antonio Pescapè, and Hyun chul Kim. Traffic classification through joint distributions of packet-level statistics. In *GLOBECOM*, pages 1–6, 2011.
19. Alberto Dainotti, Antonio Pescapé, and K.C. Claffy. Issues and future directions in traffic classification. *Network, IEEE*, 26(1):35 –40, january-february 2012.
20. Alberto Dainotti, Antonio Pescapè, and Carlo Sansone. Early classification of network traffic through multi-classification. In *TMA*, pages 122–135, 2011.
21. Alberto Dainotti, Antonio Pescapè, Carlo Sansone, and Antonio Quintavalle. Using a behaviour knowledge space approach for detecting unknown ip traffic flows. In *MCS*, pages 360–369, 2011.
22. Pedro M. Santiago del Ro, Dario Rossi, Francesco Gringoli, Lorenzo Nava, Luca Salgarelli, and Javier Aracil. Wire-speed statistical classification of network traffic on commodity hardware. In ACM IMC 2012.
23. Jeffrey Erman, Martin Arlitt, and Anirban Mahanti. Traffic classification using clustering algorithms. In *MineNet '06: Mining network data (MineNet) Workshop at ACM SIGCOMM '06*, Pisa, Italy, 2006.

24. Jeffrey Erman, Anirban Mahanti, Martin Arlitt, and Carey Williamson. Identifying and discriminating between web and peer-to-peer traffic in the network core. In *Proceedings of the 16th international conference on World Wide Web*, WWW '07, pages 883–892, Banff, Alberta, Canada, 2007.
25. A. Finamore, M. Mellia, M. Meo, and D. Rossi. Kiss: Stochastic packet inspection. In *Traffic Measurement and Analysis (TMA), Springer-Verlag LNCS 5537*, pages 117–125, May 2009.
26. Alessandro Finamore, Marco Mellia, Michela Meo, and Dario Rossi. Kiss: Stochastic packet inspection classifier for udp traffic. *IEEE/ACM Transaction on Networking*, 18(5):1505–1515, 2010.
27. Alessandro Finamore, Michela Meo, Dario Rossi, and Silvio Valenti. Kiss to Abacus: A Comparison of P2P-TV Traffic Classifiers. In *Traffic Monitoring and Analysis, Springer Lecture Notes in Computer Science*, volume 6003, pages 115–126. 2010.
28. Tom Z. J. Fu, Yan Hu, Xingang Shi, Dah-Ming Chiu, and John C. S. Lui. PBS: Periodic Behavioral Spectrum of P2P Applications. In *Proc. of PAM '09*, Seoul, South Korea, Apr 2009.
29. F. Gringoli, Luca Salgarelli, M. Dusi, N. Cascarano, F. Risso, and k. c. claffy. GT: picking up the truth from the ground for internet traffic. *ACM SIGCOMM Comput. Commun. Rev.*, 39(5):12–18, 2009.
30. Patrick Haffner, Subhabrata Sen, Oliver Spatscheck, and Dongmei Wang. ACAS: automated construction of application signatures. In *ACM SIGCOMM Workshop on Mining Network Data (Minenet'05)*, Philadelphia, PA, August 2005.
31. Marios Iliofotou, Prashanth Pappu, Michalis Faloutsos, Michael Mitzenmacher, Sumeet Singh, and George Varghese. Network monitoring using traffic dispersion graphs (tdgs). In *Proc. of IMC '07*, San Diego, California, USA, 2007.
32. M. Jamshed, J. Lee, S. Moon, I. Yun, D. Kim, S. Lee, Y. Yi, and K.S. Park. Kargus: a highly-scalable software-based intrusion detection system. 2012.
33. Yu Jin, Nick Duffield, Patrick Haffner, Subhabrata Sen, and Zhi-Li Zhang. Inferring applications at the network layer using collective traffic statistics. *SIGMETRICS Perform. Eval. Rev.*, 38, June 2010.
34. T. Karagiannis, A. Broido, N. Brownlee, kc klaffy, and M. Faloutsos. Is P2P dying or just hiding? In *IEEE GLOBECOM '04.*, Dallas, Texas, US, 2004.
35. Thomas Karagiannis, Andre Broido, Michalis Faloutsos, and Kc claffy. Transport layer identification of P2P traffic. In *4th ACM SIGCOMM Internet Measurement Conference (IMC'04)*, Taormina, IT, October 2004.
36. Thomas Karagiannis, Konstantina Papagiannaki, Nina Taft, and Michalis Faloutsos. Profiling the end host. In *Proceedings of the 8th international conference on Passive and active network measurement*, PAM'07, Louvain-la-Neuve, Belgium, 2007.
37. Amir R. Khakpour and Alex X. Liu. High-speed flow nature identification. In *Proceedings of the 2009 29th IEEE International Conference on Distributed Computing Systems*, ICDCS '09, 2009.
38. H. Kim, K. Claffy, M. Fomenkov, D. Barman, M. Faloutsos, and K. Lee. Internet traffic classification demystified: myths, caveats, and the best practices. In *Proc. of ACM CoNEXT 2008*, Madrid, Spain, 2008.
39. Ron Kohavi and Ross Quinlan. Decision tree discovery. In *IN HANDBOOK OF DATA MINING AND KNOWLEDGE DISCOVERY*, pages 267–276. University Press, 1999.
40. S. B. Kotsiantis. Supervised machine learning: A review of classification techniques. In *Proceeding of the 2007 conference on Emerging Artificial Intelligence Applications in Computer Engineering: Real Word AI Systems with Applications in eHealth, HCI, Information Retrieval and Pervasive Technologies*, pages 3–24, Amsterdam, The Netherlands, The Netherlands, 2007. IOS Press.

41. Sailesh Kumar and Patrick Crowley. Algorithms to accelerate multiple regular expressions matching for deep packet inspection. In *In Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication (SIGCOMM06)*, pages 339–350, 2006.
42. W. Li, M. Canini, A. W. Moore, and R. Bolla. Efficient application identification and the temporal and spatial stability of classification schema. *Computer Networks*, 53(6):790–809, 2009.
43. Y. Liu, D. Xu, L. Sun, and D. Liu. Accurate traffic classification with multi-threaded processors. In *IEEE International Symposium on Knowledge Acquisition and Modeling Workshop (KAM) 2008.*
44. Justin Ma, Kirill Levchenko, Christian Kreibich, Stefan Savage, and Geoffrey M. Voelker. Unexpected means of protocol inference. In *6th ACM SIGCOMM Internet Measurement Conference (IMC'06)*, Rio de Janeiro, BR, October 2006.
45. Anthony Mcgregor, Mark Hall, Perry Lorier, and James Brunskill. Flow clustering using machine learning techniques. In *PAM'04*, Antibes Juan-les-Pins, Fr., April 2004.
46. Marco Mellia, Antonio Pescapè, and Luca Salgarelli. Traffic classification and its applications to modern networks. *Computer Networks*, 53(6):759–760, 2009.
47. A. Moore, D. Zuev, and M. Crogan. Discriminators for use in flow-based classification. Technical report, University of Cambridge, 2005.
48. Andrew W. Moore and Denis Zuev. Internet traffic classification using bayesian analysis techniques. In *ACM SIGMETRICS '05*, Banff, Alberta, Canada, 2005.
49. David Moore, Ken Keys, Ryan Koga, Edouard Lagache, and K. C. Claffy. The coralreef software suite as a tool for system and network administrators. In *Proceedings of the 15th USENIX conference on System administration*, San Diego, California, 2001.
50. 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.
51. Napa-Wine. `http://www.napa-wine.eu/`.
52. T. T. T. Nguyen and G. Armitage. A survey of techniques for internet traffic classification using machine learning. *IEEE Communications Surveys & Tutorials*, 10(4):56–76, 2008.
53. Vern Paxson. Bro: a system for detecting network intruders in real-time. *Elsevier Comput. Netw.*, 31:2435–2463, December 1999.
54. F. Risso, M. Baldi, O. Morandi, A. Baldini, and P. Monclus. Lightweight, payload-based traffic classification: An experimental evaluation. In *Proc. of IEEE ICC '08*, May 2008.
55. F. Risso and N. Cascarano. Diffinder available at `http://netgroup.polito.it/research-projects/l7-traffic-classification`.
56. Martin Roesch. Snort - lightweight intrusion detection for networks. In *Proceedings of the 13th USENIX conference on System administration*, LISA '99, pages 229–238, Berkeley, CA, USA, 1999. USENIX Association.
57. Dario Rossi and Silvio Valenti. Fine-grained traffic classification with Netflow data. In *TRaffic Analysis and Classification (TRAC) Workshop at IWCMC '10*, Caen, France, Jun 2010.
58. Matthew Roughan, Subhabrata Sen, Oliver Spatscheck, and Nick Duffield. Class-of-service mapping for QoS: a statistical signature-based approach to IP traffic classification. In *ACM SIGCOMM Internet Measurement Conference (IMC'04)*, Taormina, IT, October 2004.
59. L. Salgarelli, F. Gringoli, and T. Karagiannis. Comparing traffic classifiers. *ACM SIGCOMM Comp. Comm. Rev.*, 37(3):65–68, 2007.
60. Subhabrata Sen, Oliver Spatscheck, and Dongmei Wang. Accurate, scalable in-network identification of p2p traffic using application signatures. In *13th international conference on World Wide Web (WWW'04)*, New York, NY, US, May 2004.

61. Yeon sup Lim, Hyunchul Kim, Jiwoong Jeong, Chong kwon Kim, Ted Taekyoung Kwon, and Yanghee Choi. Internet traffic classification demystified: on the sources of the discriminative power. In *CoNEXT*, page 9, 2010.
62. G. Szabó, I. Gódor, A. Veres, S. Malomsoky, and S. Molnár. Traffic classification over Gbit speed with commodity hardware. *IEEE J. Communications Software and Systems*, 5, 2010.
63. Silvio Valenti, Dario Rossi, Michela Meo, Marco Mellia, and Paola Bermolen. Accurate, Fine-Grained Classification of P2P-TV Applications by Simply Counting Packets. In *Proc. of International Workshop on Traffic Monitoring and Analysis (TMA '09), Springer Lecture Notes on Computer Science*, volume 5537, pages 84–92, Aachen, Germany, 2009.
64. Giorgos Vasiliadis, Michalis Polychronakis, and Sotiris Ioannidis. Midea: a multi-parallel intrusion detection architecture. In *ACM Conference on Computer and Communications Security*, pages 297–308, 2011.
65. N. Williams, S. Zander, and G. Armitage. A preliminary performance comparison of five machine learning algorithms for practical IP traffic flow classification. *ACM SIGCOMM CCR*, 36(5):5–16, 2006.
66. Wm. A. Wulf and Sally A. Mckee. Hitting the memory wall: Implications of the obvious. *Computer Architecture News*, 23:20–24, 1995.
67. Kuai Xu, Zhi-Li Zhang, and Supratik Bhattacharyya. Profiling internet backbone traffic: behavior models and applications. *ACM SIGCOMM Comput. Commun. Rev.*, 35(4):169–180, 2005.
68. Yuan Zu, Ming Yang, Zhonghu Xu, Lin Wang, Xin Tian, Kunyang Peng, and Qunfeng Dong. Gpu-based nfa implementation for memory efficient high speed regular expression matching. In *PPOPP*, pages 129–140, 2012.