Multi-Classification Approaches for Classifying Mobile App Traffic

Giuseppe Aceto[a,b], Domenico Ciuonzo[b], Antonio Montieri[a], Antonio Pescapé[a,b]

[a] *University of Napoli "Federico II", Italy*
[b] *Network Measurement and Monitoring (NM2) s.r.l., Italy*

Abstract

The growing usage of smartphones in everyday life is deeply (and rapidly) changing the nature of traffic traversing home and enterprise networks, and the Internet. Different tools and middleboxes, such as performance enhancement proxies, network monitors and policy enforcement devices, base their functions on the knowledge of the applications generating the traffic. This requirement is tightly coupled to an accurate traffic classification, being exacerbated by the (daily) expanding set of apps and the moving-target nature of mobile traffic. On the top of that, the increasing adoption of encrypted protocols (such as TLS) makes classification even more challenging, defeating established approaches (e.g., Deep Packet Inspection).

To this end, in this paper we aim to improve the performance of classification of mobile apps traffic by proposing a multi-classification (viz. fusion) approach, intelligently-combining outputs from state-of-the-art classifiers proposed for mobile and encrypted traffic classification. Under this framework, four classes of different combiners (differing in whether they accept soft or hard classifiers' outputs, the training requirements, and the learning philosophy) are taken into account and compared. The present approach enjoys modularity, as any classifier may be readily plugged-in/out to improve performance further. Finally, based on a dataset of (true) users' activity collected by a mobile solutions provider, our results demonstrate that classification performance can be improved according to all considered metrics, up to +9.5% (recall score) with respect to the best state-of-the-art classifier. The proposed system is also capitalized to validate a novel pre-processing of traffic traces, here developed, and assess performance sensitivity to traffic object (temporal) segmentation, before actual classification.

Keywords: traffic classification, mobile apps, Android apps, iOS apps, encrypted traffic, information fusion, classification combining, multi-classification.

1. Introduction

Several tools, such as security/quality-of-service enforcement devices and network monitors base their operations on the knowledge of the application generating the traffic. As a consequence, their use is limited (or impaired) when this requirement is not (or loosely) satisfied.

The process of associating (labeling) network traffic with specific applications or application types is known as Traffic Classification (TC) and has a long-established application in several fields, backed by a wide scientific literature [1, 2, 3, 4]. This process is increasingly challenged by recent evaluations in Internet usage, as the global spread and growing usage of smartphones is profoundly changing the kind of traffic that travels over home and enterprise networks and the Internet. Thereupon, both the necessity and the difficulty of TC of mobile traffic have become very high nowadays. Indeed, other than the traditional drivers for TC, classification of mobile apps' traffic has the potential of providing extremely valuable profiling information (e.g., to advertisers, insurance companies and security agencies). On the other hand, it surely raises privacy issues, especially in regards to recognition of context-sensitive apps (such as health and dating ones) by malicious parties. Unluckily, TC comes with its own challenges and requirements that are even exacerbated in a mobile-traffic context, usually characterized by a large number of apps to discriminate from and an inadequate number of training samples per app, which hinder the achievement of satisfactory performance. Moreover, the increasing adoption of encrypted protocols (TLS) makes the classification even more challenging, defeating established approaches.

Moving from earlier port-based methods, to those based on payload inspection (termed Deep Packet Inspection methods, DPI [5, 6]), approaches based on Machine Learning (ML) classifiers are deemed the most appropriate, especially in this context, since they suit also Encrypted Traffic (ET) analysis [7, 8, 9, 10].

Indeed, in the latter context, it is crucial resorting to the sequence of packets [7, 8, 9, 10] or message sizes [11, 12], rather than their content. Then ML techniques may be applied either directly on the whole sequence (such as in [10, 11, 12]) or based on statistics/histograms extracted from it (such as in [7, 8, 10, 12]). It is worth noting that the above statistical techniques can be also combined with port-association algorithms (in scenarios where port-info can be considered reliable) to develop hybrid approaches, such as [13]. Although earlier results have been published on this topic, the traffic of mobile apps is

Email addresses: giuseppe.aceto@unina.it (Giuseppe Aceto), ciuonzo@nm-2.com (Domenico Ciuonzo), antonio.montieri@unina.it (Antonio Montieri), pescape@unina.it (Antonio Pescapé)

a *moving target* for classifiers due to its dynamic evolution and mix. Thus mobile TC constitutes an open and evolving research field.

In this paper, we aim to improve the classification performance of mobile apps by proposing a *Multi-Classification System* (MCS) which intelligently-combines decisions from state-of-the-art (base) classifiers specifically devised for mobile- and encrypted-traffic classification and currently considered the best approaches in such context [7, 8, 10]. The proposed MCS is graphically depicted as a whole in Fig. 1. To the best of authors' knowledge, this investigation is performed in the mobile context for the first time[1]. Additionally, despite (wise) combination of state-of-the-art classifiers is here analyzed to show how current classification performance of mobile traffic can be improved, the proposed MCS is nor restricted to the considered set of classification algorithms and statistical features, neither to the operational scenario (i.e. classifiers for "early" TC [15, 16] may be considered in the proposed framework without any further complication [17, 18]). Indeed, the MCS framework can potentially overcome the deficiencies of each single classifier (not improvable over a certain bound, despite efforts in careful "tuning") and provide improved performance w.r.t. any of the base classifiers, also allowing for *modularity* of classifiers' selection in the pool. For this reason, research has focused on MCSs in the last years [19, 20, 21, 22].

Additionally, with respect to the aforementioned works, our MCS allows for choosing from several types of combiners (based on both hard and soft approaches, the latter successfully applied to many practical problems [23] and whose application to mobile TC is deemed extremely appealing) developed in the literature [23, 24] constituting a wide spectrum of achievable performance, operational complexity, and training set requirements. The generality and the weak-coupling to any base classifier of the proposed MCS is also capitalized to draw out "best practices" in mobile traces' pre-processing and (proper) traffic object segmentation.

Based on a dataset collected by a global mobile solutions provider[2] of true users' activity, our results show that MCS framework can improve classification performance with respect to the best base classifiers considered for the task. Specifically, it is shown that macro recall can be appealingly improved by more than +9% on the best base classifier, and that there is room for further possible improvement with evidence of over +10% achievable by the ideal combiner. Finally, an investigation of subset selection of classifiers' pool (referring to all the combiners within the proposed MCS), is also reported, highlighting an additional path of improvement (and complexity reduction).

The paper is organized as follows. Sec. 2 discusses related works, whereas Secs. 3–5 collectively describe the considered MCS for mobile TC. More specifically, Sec. 3 introduces the classification objects and the employed set of features, whereas Sec. 4 describes the classification algorithms (considered as base classifiers). Then, Sec. 5 introduces the (hard and soft) fusion techniques adopted for their combination. Such detailed description is aimed at the full specification of the present approach, so as to enable easy implementation or porting to any architecture, and comparison with other approaches and tools. Experimental results are reported in Sec. 6. Finally, Sec. 7 provides conclusions and future directions.

2. State-of-the-art Techniques for Traffic Classification of Mobile Apps

TC of mobile apps has been object of huge interest by several recent works, mainly based on ET assumption. Dai et al. [25] first introduced the concept of "network profile", playing the same role as DNA profiles for an Android app (i.e. a network fingerprint). They proposed *NetworkProfiler*, a system composed of a module automatically executing an app in an emulator (*DroidDriver*), and another module that from the generated network traffic builds a profile in terms of (*i*) contacted hosts and (*ii*) a state machine of string sequences in URLs (*Fingerprint Extractor*). Being based on DPI (e.g., HTTP payload) features, the extractor *is not suited* for ET. The approach has been shown to be effective in identifying ad-traffic, whereas for non-ad apps the evaluation has been carried out only for 6 apps. Additionally, in [25] the full ground truth of the traffic traces being analyzed is not available, so making it hard to quantify the classification performance of *NetworkProfiler*. A similar spirit permeates the review of Tongaonkar [26], where challenges and techniques for mobile TC and app identification are discussed, mainly based on signature generation and fingerprint extraction from mobile traffic payloads and apps' metadata, as well as from 3rd-party services (e.g., advertisement and profiling traffic). Nevertheless, the problem of dissecting ET is there *bypassed* by considering man-in-the-middle solutions, suitable only in controlled environments such as enterprises.

Stöber et al. [27] developed a fingerprinting scheme for devices by learning their traffic patterns through background activities. They contend that 70% of smartphone traffic belongs to background activities, and this can be leveraged to create a fingerprint. Based on 3G transmissions, *bursts* of data are considered to evaluate statistical features. Then, by means of Support Vector Classifier (SVC) and K-Nearest Neighbors, a model of the traffic to be fingerprinted is built, being capable of identifying similar bursts. Results show that using $\approx$ 15 minutes of traffic testing (based on 6 hours of training) leads to an accuracy $\geq$ 90% (among 20 users with different combinations of apps installed). Wang et al. [28] propose a system for classifying app usage over encrypted 802.11 traffic (reporting results for 13 iOS apps from 8 distinct categories). Data frames are collected from target apps by running them dynamically for 5 minutes and training a Random Forest (RF) classifier with the proposed set of features. The need for an accurate ground-truth labeling is raised, highlighted by a counterintuitive behavior of some app performance with the training time. *AppScanner* is proposed in [10] as a framework for fingerprinting and identification of mobile apps. The fingerprints are collected by running apps *automatically* on an Android device and the network

[1]Preliminary results in the same framework of this study have been accepted as a conference publication and will be published as [14].

[2]Due to NDA with the provider we can not report its name, details of its network, detailed information on the data set, nor release the data set.

traces are pre-processed (to remove background traffic and extract features) to train an SVC and an RF. Statistical features are collected on sets of packets defined through timing criteria and destination IP address/port (see Sec. 3.1). The results, evaluated on 110 most popular apps from Google Play Store, report 99% average accuracy in identifying single apps, and up to 86.9% in classifying them, *outperforming* state-of-the-art alternatives devised for the (conceptually-)similar website fingerprinting issue [7, 8]. More recently, AppScanner has been employed on a larger dataset to test the aging of apps' fingerprints (due to updates) and possible invariance with respect to used device and app versions (due to different users' usage) [29]. It is demonstrated that, though updates, time, and different devices lead to a performance degradation (with updates being the more demanding issue), a good classification accuracy can be still achieved. To this end, a method for the removal of background / 3rd-party services traffic is there conceived, however not verified by an accurate labeling of the actual non-specific app traffic. The terms of comparison in [10, 29] are also used by Alan and Kaur [30] to investigate whether Android apps can be identified from their launch-time traffic using only TCP/IP headers (i.e. the sizes of the first 64 packets). They find that apps can be identified with 88% accuracy when training and test sets are collected on the same device, based on the simple classification methods developed in [7, 8]. On the other hand, accuracy drops significantly (up to 26% for the best classifier) when the OS/vendor is different. The same work analyzes the impact of the amount of training data required for classification and its "aging" (due to updates). It is worth noticing that state-of-the-art approaches [10, 31, 32] considered in this work outperform those analyzed in [30] also in terms of other performance metrics (see Sec. 6).

Other works aimed at identifying fine-grained user actions within mobile-app traffic. Conti et al. [33] recognizes specific actions that users perform while running a certain app, based on packet direction/size info. This is achieved through *service burst* (see Sec. 3.1) classification via RF approach, leading to $\geq$ 95% accuracy for most of the considered actions within a set of 7 Android apps. *Netscope* [9] performs a similar task taking into account a set of 35 different activities (for both iOS and Android devices), based on statistics originated from IP headers. Assuming an eavesdropper on a Wi-Fi network, it is shown that even a small portion of ET is enough for a given app to be recognized. K-means clustering is employed for elementary-behavior discovery and then an SVC is trained/tested on activity-behaviors binary mapping, showing performance that varies with the device being tested, but reach 78.04% precision and 76.04% recall, on average.

Although not focused on mobile apps (but readily adaptable to this context), the work in [31] proposes a technique to precisely identify services running within HTTPS connections, without relying on specific header fields (being prone to alteration). Suitable features for HTTPS traffic are defined and used as input for a ML-based multi-level identification framework. The evaluation, based on real traffic, shows high identifiability of encrypted web services. Finally, a related work focusing on ET classification, is presented in [34], where the SSL/TLS-state fingerprint sequence is modelled as a second-order Markov chain, jointly with a bigram-attribute clustering of two relevant features, to obtain satisfactory accuracy results in a pool with 14 applications, all containing a high rate of ET.

3. Traffic Classification Definitions and Features

In this section we introduce terms and concepts regarding traffic objects, along with the definition of the features extracted from observed traffic and adopted for classification.

3.1. Traffic View

A common TC object is the *biflow*, defined as a sequence of packets sharing the values of the 5-ple *(transport protocol, source IP address & transport port, destination IP address & transport port)*, where source and destination can be *swapped* [2]. On the other hand, in this paper, network traffic is decomposed into *service bursts* (SBs), leveraging the notions introduced in [27] and [10, 33] for mobile-phone identification and mobile-app classification, respectively.

To this end, we provide the preliminary definition of *burst* [27], being a sequence of packets having an inter-packet time smaller than a given threshold (named *Burst Threshold*, BT), irrespective of their source or destination addresses, as well as of the biflow they belong to. Accordingly, a SB is then a set of packets, within a single burst, that belongs to biflows sharing the same transport protocol, destination IP address & port number.[3]

It is worth noting that the BT is a key parameter in the definition of SBs and a few different values have been chosen in recent studies [35, 10, 29]. This study also accounts for sensitivity of classification performance to this parameter (see Sec. 6.3). The process of extracting the SBs from the considered traffic traces is summarized in Fig. 1 through the block **SB Extraction**. In the following of the paper (precisely in Sec. 6.3) we will also investigate the need for a preprocessing step (represented in Fig. 1 as the **Preprocess** block) and, in affirmative case, whether this should be performed before or after burstification process.

Remark: We point that SB notion has been used previously in [10, 33] under the (different) name of *flow*. However, in this paper, to avoid any ambiguity with the common and established definition of flow [2], we will refer to the decomposition used in [10, 33] as a SB.

3.2. Statistical Features

For the purpose of TC, we will consider features which are extracted (by statistical means) from the (whole) vector of packet lengths of the generic SB. This approach is analogous to flow-based TC when a flow (resp. a biflow) is instead considered as the relevant object of classification and features are

[3]The biflow direction is defined according to its first packet: the packet source (destination) is chosen as source (destination) for the whole biflow. Criteria and heuristics for biflow start and end can be defined for both TCP and UDP, and in general a TCP biflow does *not* necessarily match with a TCP session (see [2]).

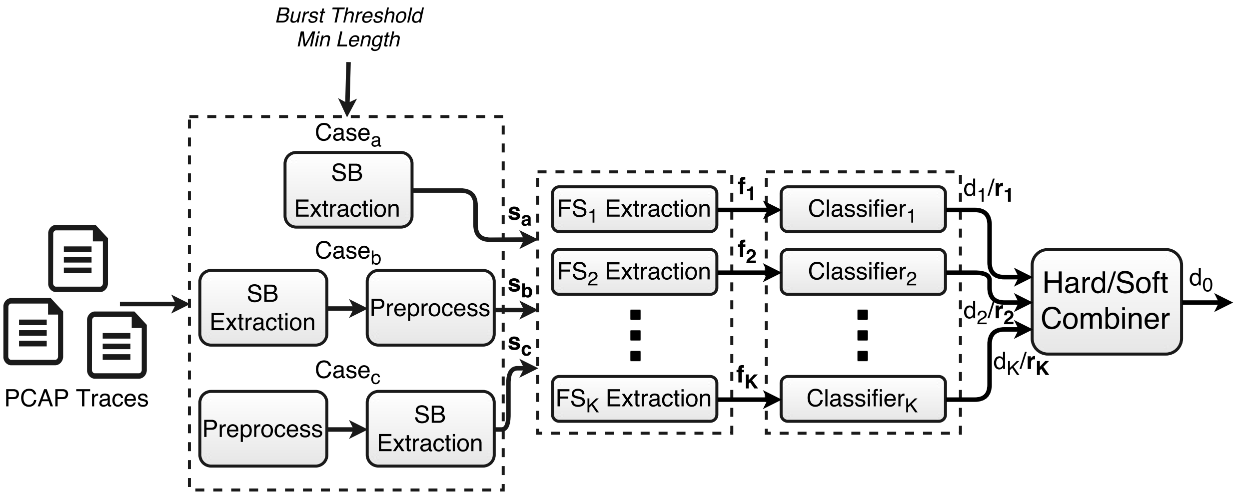


Figure 1: Architecture of the Multi-Classification System (MCS) proposed.

extracted from the sequence of packets forming it. It is worth mentioning that other feature sets may be considered, especially when early-TC of the generic SB is deemed of interest. The aforementioned class includes the packet sizes of the first K packets or some statistical features extracted from this "early" segment, as studied in [17, 18] for the case of Internet TC.

For each SB, three packet series are here considered: (*i*) *incoming* packets only (In), (*ii*) *outgoing* packets only (Out), and (*iii*) *bidirectional* traffic (i.e. both incoming and outgoing packets, In&Out). The following features can be identified for each of these series [10]:

- vector of packet lengths with sign indicating direction;
- minimum, maximum, mean, median, absolute deviation, standard deviation, variance, skew, and kurtosis;
- percentiles (from 10% to 90%, with 10% increments).

Also, for the incoming and outgoing packet series taken as a whole, the joint histogram of packet lengths in both directions can be considered [7, 8].

Finally, in the following, the set of M features adopted by each classifier will be generically indicated with $f_1, \ldots, f_M$ (or collectively as $\boldsymbol{f} \triangleq \begin{bmatrix} f_1 & \cdots & f_M \end{bmatrix}^T$) and the set of classes (apps) as $\Omega \triangleq \{c_1, \ldots, c_L\}$.

The process of extracting the feature set for k^{th} classifier from each SB is summarized in Fig. 1 through the block **FS**$_k$ **Extraction**.

4. Classification Algorithms

In this section we list the state-of-art approaches that we selected as the pool of $K = 9$ *base* classifiers employed in our MCS. The generic k^{th} (base) classifier within the considered pool is represented in Fig. 1 by means of the block **Classifier**$_k$. More specifically, this block *receives* as input the corresponding k^{th} *feature set* (from the preceding feature extraction block) and *outputs* either a *hard or soft decision* (mathematical details are later provided in Sec. 5) to the hard/soft combiner. We briefly describe their main properties and the motivations that guided us to their choice. For all of them we have reproduced their exact implementation and executed them with the same parameters as described in the respective works, to which we refer for further details.

A recap of the base classifiers considered in this paper, along with the abbreviations used, the supervised philosophy, the set of features taken as input, and the corresponding reference is given in Tab. 1.

Lib_NB

In Liberatore and Levine [8], two classifiers were proposed, one based on the Jaccard similarity index and another based on the Naïve Bayes (NB) learning technique. It was observed that the NB enjoys attractive performance and increased robustness than the Jaccard-based classifier, if IP packets are padded; thus we select NB-based approach as a base classifier (`Lib_NB`). The NB assumes class-conditional independence of the features $\boldsymbol{f}$ (being not the case for real-world problems but working well in practice) and evaluates the probability that a test instance $\boldsymbol{f}_T$ belongs to each class c_i, i.e. the posterior probability $P(c_i|\boldsymbol{f}_T)$ through the Bayes' theorem $P(c_i|\boldsymbol{f}_T) \propto P(c_i) \prod_{m=1}^{M} P(f_{T,m}|c_i)$, where "$\propto$" denotes proportionality. Term $P(c_i)$ denotes the (prior) probability that a generic sample from the dataset will belong to c_i and is estimated from the training set population, while each PDF $P(f_{T,m}|c_i)$ is estimated by employing (Gaussian) kernel density estimation. The fine-grained feature there employed is the joint histogram of packet lengths in both incoming and outgoing directions.

Her_Pure, Her_TF, and Her_Cos

Herrmann et al. [7] proposed the use of a Multinomial NB (MNB) classifier, adopting the *same* set of features as `Lib_NB` [8], but *differing* in the building assumption. Indeed, while the NB classifier estimates each feature PDF using Gaussian kernels whose occurrence frequencies of the various packet

Table 1: Summary of state-of-art techniques selected as base classifiers.

Abbreviation	**Method**	**Features set**	**Reference**
`Lib_NB`	Naïve Bayes (NB)	Joint In&Out Histogram	Liberatore and Levine [8]
`Her_Pure/TF/Cos`	Multinomial NB	Joint In&Out Histogram	Herrmann et al. [7]
`Tay_RF`	Random Forest	Stat. + Percent. (In/Out/In&Out)	Taylor et al. [10]
`Tay_SVC`	Support Vector Classifier	Stat. + Percent. (In/Out/In&Out)	Taylor et al. [10]
`CART`	Decision Tree	Stat. + Percent. (In/Out/In&Out)	Bakhshi and Ghita [32]

sizes match best with the observed values in the test instance, the MNB classifier treats the f_ms as frequencies of a certain value of a categorical random variable and compares the sample histogram of each test instance with the aggregated histogram of all training instances per class. Then, the evaluation of the conditional PMF $P(\boldsymbol{f}_T|c_i)$ is different from `Lib_NB` and equals $P(\boldsymbol{f}_T|c_i) \propto \prod_{m=1}^{M}(\rho_m)^{f_{T,m}}$, where ρ_m denotes the probability of sampling the m^{th} feature. This implementation is referred to as `Her_Pure` in our analysis. A few variants of MNB classifier, adopting *term frequency transformation* without and with cosine normalization, were also successfully employed in [7] and compared in [10], and are referred in our analysis to as `Her_TF` and `Her_Cos`, respectively.

`Tay_RF` and `Tay_SVC`

In Taylor et al. [10], four (resp. two) approaches for mobile-app traffic classification (resp. identification) were proposed, leveraging both an SVC and an RF. An SVC is a supervised model that represents the training samples as points in a feature (vector) space, with the aim of finding a set of hyperplanes which provide the best class separation. Then, during testing phase, the SVC classifies new points according to the portion of space they fall into. On the other hand, an RF is an ensemble classification method taking advantage of several decision trees (obtained by combining the ideas of *bootstrap aggregating* and *random-feature selection* to avoid over-fitting) built at training time in order to form a stronger classifier [36].

In [10], these classifiers were fed with either (*i*) raw vectors of packet lengths or (*ii*) statistical features (i.e. statistics and percentiles pertaining to incoming/outcoming/bidirectional packet sequences) traffic, with the latter approach leading to the best and least complex classifier (RF with statistical features) between the two. The latter set has been drawn out in [10] as the most "informative" from a larger set of 54 statistical features by means of feature selection technique on mobile traffic data. For this reason, we consider both RF (`Tay_RF`) and SVC (`Tay_SVC`) based on the 40 statistical features selected in [10].

`CART`

Several works performed TC by means of decision trees (e.g., C4.5, C5.0, and their variants), both as flat classifiers [37, 32] and also in a hierarchical [31] or multi-classification [19] architecture. In this paper, we leverage Classification And Regression Tree (`CART`), a very similar variant of the C4.5 algorithm, constructing binary trees exploiting the features and thresholds that ensure the maximum information gain at each node and allowing to perform both classification and regression tasks (i.e. with categorical and numerical target variables, respectively). The above classifier is fed with the same statistical features as `Tay_RF` and `Tay_SVC`.

5. Classifier Fusion Techniques

Different classifier fusion rules (viz. combiners) have been proposed in the literature [19, 23]. In this section, we will focus on *hard combiners* first (Sec. 5.1), relying on *Type 1 classifiers* (i.e. those that output only the predicted class). Then, we will discuss fusion rules resorting to classifiers' soft-outputs (viz. *Type 3 classifiers*), namely the *soft combiners* (Sec. 5.2). The generic (hard/soft) combiner adopted within the proposed MCS is shown in Fig. 1 through the block **Hard/Soft Combiner**.

In the proposed MCS we will consider both *non-trainable* and *trainable* combiners [23]. In the former case, the combiner has no extra parameters that need to be trained (the combiner is ready-to-use once the sole base classifiers are trained). In the latter case, the combiner requires some parameters to be estimated, usually by means of a *validation set*, different from both the training and the test sets. Overall, the proposed MCS will provide *twenty different choices* (6 hard- and 14 soft-combiners, respectively) as the classifier-fusion block being employed.

Finally, for completeness of performance evaluation, in Sec. 6.4, we will also consider an ORAcle combiner (`ORA`), i.e. an ideal upper bound on the performance corresponding to a combiner correctly classifying a test sample if *at least one* of the base classifiers provides the correct decision [23].

5.1. Hard Combiners

Hard combiners are based on Type 1 classifiers, that is they exploit only the classifiers' predicted classes (generically denoted with $\hat{d}_k(\boldsymbol{f})$ and collectively as $\hat{\boldsymbol{d}}(\boldsymbol{f}) \triangleq \begin{bmatrix}\hat{d}_1(\boldsymbol{f}) & \cdots & \hat{d}_K(\boldsymbol{f})\end{bmatrix}^T$), implying the least requirements for designers [23].

In what follows, we will denote with $\mu_i(\hat{\boldsymbol{d}}_T)$ the confidence attributed to the i^{th} class by a generic hard combiner, based on decisions $\hat{\boldsymbol{d}}_T \triangleq \hat{\boldsymbol{d}}(\boldsymbol{f}_T)$ pertaining to the test instance $\boldsymbol{f}_T$.

Then, the combiner decision is obtained as

$$\hat{d}_0 \triangleq \arg\max_{i\in\Omega} \mu_i(\hat{\boldsymbol{d}}_T).$$

Before proceeding, we recall the definition of k^{th} classifier confusion matrix $\boldsymbol{E}^k$ [23], whose $(i, j)^{th}$ entry is denoted with $e^K_{i,j}$ and represents the probability of k^{th} classifier deciding for j^{th}

class when the i^{th} class is being observed. Clearly, the matrices $\boldsymbol{E}_k$ (as well as the priors $P(c_i)$) employed by combiners are typically estimated using a validation set.

In this work, the following hard combiners[4] will be considered:

1. *Majority Voting* (`MV`): the estimated class corresponds to the one voted by the relative majority of the classifiers.[5]

2. *Weighted Majority Voting* (`WMV`): this approach is obtained by weighting the vote of each classifier by its relative confidence. The i^{th} class confidence of the combiner is evaluated as

$$\mu_i(\hat{\boldsymbol{d}}_T) \triangleq \left\{ \delta_i + |I_+^i| \cdot \ln(L-1) + \sum_{k \in I_+^i} w_k \right\},$$

   where I_+^i denotes the subset of classifiers having decided for i^{th} class, $\delta_i \triangleq [\ln P(c_i)]$ denotes a class-constant offset, and $w_k \triangleq \ln(p_k/(1-p_k))$ denotes the weight of k^{th} classifier, p_k being the (estimated) accuracy [24].

3. *Recall Combiner* (`REC`): this combiner relaxes the assumption of equal class-conditional accuracy (viz. recall) in `WMV` and thus it amounts to different individual class-specific recalls. The `REC` confidence measure is then

$$\mu_i(\hat{\boldsymbol{d}}_T) \triangleq \left\{ \bar{\delta}_i + |I_+^i| \cdot \ln(L-1) + \sum_{k \in I_+^i} w_{k,i} \right\},$$

   where I_+^i denotes the subset of classifiers having decided for i^{th} class, $\bar{\delta}_i \triangleq [\ln P(c_i) + \sum_{k=1}^{K} \ln(1-p_{k,i})]$ denotes a class-constant offset, and $w_{k,i} \triangleq \ln(p_{k,i}/(1-p_{k,i}))$ denotes the weight of k^{th} classifier when deciding for i^{th} class, $p_{k,i}$ being its (estimated) class-conditional accuracy [24].

4. *Naïve Bayes* (`NB`): the i^{th} class confidence measure is represented by the *a posteriori* probability $P(c_i|\hat{d}_1, \ldots, \hat{d}_K)$ based on the conditional independence of classifiers, that is

$$\mu_i(\hat{\boldsymbol{d}}_T) \triangleq P(c_i) \left\{ \prod_{k=1}^{K} P(\hat{d}_{k,T}|c_i) \right\}.$$

5. *Behavior-Knowledge Space method* (`BKS`): this approach removes the conditional independence assumption of `NB` combiner via multinomial counting on the joint classifiers' space $\hat{d}_1, \ldots, \hat{d}_K$ [38]. More specifically, the validation set is used to estimate the *a posteriori* probability $P(c_i|\hat{\boldsymbol{d}})$ for each c_i and for *each value* of $\hat{\boldsymbol{d}}$.[6] This allows labeling each possible value of $\hat{\boldsymbol{d}}_T$ with the most likely class, according to $\mu_i(\hat{\boldsymbol{d}}_T) \triangleq P(c_i|\hat{\boldsymbol{d}}_T)$ and constructing a look-up (BKS) table. Then, during the testing phase, each new $\hat{\boldsymbol{d}}_T$ provides an index to retrieve from BKS table the estimated class[7] $\hat{d}_0$.

6. *WERnecke's method* (`WER`): `WER` constructs the same table as `BKS` but, to reduce over-fitting, considers the 95% confidence intervals of the frequencies (calculated by adopting the normal approximation of the Binomial distribution) in each unit [23]. If there is overlap among the intervals, there is no dominating class for labeling the test instance $\hat{\boldsymbol{d}}_T$. In this case, the "least wrong" among the K classifiers is identified (based on confusion matrices) and authorized to assign the class to that unit.

5.2. Soft Combiners

This section discusses combiners based on *Type 3 classifiers*. More specifically, we assume that k^{th} classifier is able to provide a soft-output vector $\boldsymbol{r}_k(\boldsymbol{f})$ collecting L degrees of support (each belonging[8] to the interval $[0, 1]$), whose i^{th} entry $d_{k,i}(\boldsymbol{f})$ denotes the confidence that k^{th} classifier gives to the hypothesis that $\boldsymbol{f}$ was generated from class c_i. Consequently, for a feature vector input $\boldsymbol{f}$ the outputs of a pool of K classifiers can be summarized in a $K \times L$ Decision Profile (DP) matrix, denoted with $\boldsymbol{D}(\boldsymbol{f})$. It is worth noting that k^{th} row of $\boldsymbol{D}(\boldsymbol{f})$ equals $\boldsymbol{r}_k(\boldsymbol{f})$, whereas i^{th} column of $\boldsymbol{D}(\boldsymbol{f})$, denoted with $\boldsymbol{d}_i(\boldsymbol{f})$, represents the soft-confidence attributed to i^{th} class by the classifiers' pool.

In what follows, we will denote with $\mu_i(\boldsymbol{D}(\boldsymbol{f}_T))$ the confidence attributed to i^{th} class by the generic soft combiner based on DP matrix $\boldsymbol{D}(\boldsymbol{f}_T)$ obtained from the test instance $\boldsymbol{f}_T$. The corresponding decision is then found as:

$$\hat{d}_0 \triangleq \arg\max_{i \in \Omega} \mu_i(\boldsymbol{D}(\boldsymbol{f}_T)).$$

Soft-combiners can be mainly categorized into *Class-Conscious* (CC) and *Class-Indifferent* (CI) methods. CC methods use DP matrix but disregard part of the information, using only *one column per class* (i.e. $\mu_i(\boldsymbol{D}(\boldsymbol{f}_T)) = \mu_i(\boldsymbol{d}_i(\boldsymbol{f}_T))$). For this class of soft combiners, there exist either trainable or non-trainable combiners. On the other hand, CI methods use the whole DP matrix $\boldsymbol{D}(\boldsymbol{f}_T)$ to evaluate i^{th} class confidence, i.e. they interpret the DP as a vector in the intermediate feature space. Only trainable combiners belong to CI category.

The following soft combiners have been considered in this work:

1. *(CC) Non-trainable combiners*: the combination function can be chosen among different simple alternatives, such as the (*i*) `Mean` ($\mu_i(\boldsymbol{d}_i(\boldsymbol{f}_T)) \triangleq \frac{1}{K}\sum_{k=1}^{K} d_{k,i}(\boldsymbol{f}_T)$), the (*ii*) `Maximum` ($\mu_i(\boldsymbol{d}_i(\boldsymbol{f}_T)) \triangleq \max_k d_{k,i}(\boldsymbol{f}_T)$), (*iii*) the `Minimum` ($\mu_i(\boldsymbol{d}_i(\boldsymbol{f}_T)) \triangleq \min_k d_{k,i}(\boldsymbol{f}_T)$), and (*iv*) the `Median`

[4] Note that all the (hard) combiners are trainable, except for the Majority Voting with random tie-breaking.

[5] In case multiple classes obtain the same highest value, ties are broken either (*a*) randomly or (*b*) by using e_{ii}^k, i.e. the vote of each classifier is weighted by the confidence degree of that classifier when it assigns a sample to the class it is voting for [19]. In the latter case, `MV` becomes a *trainable combiner*.

[6] The space complexity is thus $O(L^K)$, which requires a large validation set for training.

[7] Ties are resolved by using a `MV` (with random tie-breaking) between the elements of $\hat{\boldsymbol{d}}_T$ [23].

[8] Such constraint corresponds to the natural range output of a confidence measure and can be ensured even though the specific classifier does not admit normalized soft-outputs, see [23].

($\mu_i(\boldsymbol{d}_i(\boldsymbol{f}_T)) \triangleq \mathrm{med}_k d_{k,i}(\boldsymbol{f}_T)$). Another option is ($v$) the `Trimmed (Trim) Mean`: the K degrees of support are sorted and $P\%$ of the values are dropped on both tails (this confers potential robustness to "outliers"); the $\mu_i(\boldsymbol{d}_i(\boldsymbol{f}_T))$ is found as the mean of the remaining degrees of support. Besides, we consider the `Generalized (Gen) Mean`, defined as

$$\mu_i(\boldsymbol{d}_i(\boldsymbol{f}_T)) \triangleq \left(\frac{1}{K}\sum_{k=1}^{K} d_{k,i}(\boldsymbol{f}_T)^{\alpha}\right)^{1/\alpha}$$

which comprises different means and functions as special cases [23].[9]

Finally, we consider the `Probabilistic Product (PP)` aggregation [39], providing maximum a-posteriori Bayes decision, based on the (unrealistic) assumptions that the classifiers use mutually independent subsets of features, and whose confidence measures yield the *true* posterior probability, that is $d_{k,i} = P(c_i|\hat{d}_k)$, on their respective feature subspaces. The combination formula is $\mu_i(\boldsymbol{d}_i(\boldsymbol{f}_T)) \triangleq \prod_{k=1}^{K} d_{k,i}(\boldsymbol{f}_T)/P(c_i)^{K-1}$, where the prior probabilities $P(c_i)$ are estimated from training data.

2. *(CC) Trainable combiners*: here we will consider the (i) `Fuzzy Integral` approach (`FI`) and (ii) *trainable linear combinations* [23].

   `FI` searches for the maximal grade of agreement between the objective evidence (provided by the sorted classifier outputs for i^{th} class) and the expectation (i.e. the *fuzzy measure* values). More specifically, the `FI` is based on evaluating the support as

$$\mu_i(\boldsymbol{d}_i(\boldsymbol{f}_T)) \triangleq \max_{t=1}^{K}\{\min\{d_{k_t,i}(\boldsymbol{f}_T),\ g(t)\}\}$$

   In other terms, the vector $\boldsymbol{d}_i(\boldsymbol{f}_T)$ (i.e. the values of support for c_i) is sorted in descending order and fused with the fuzzy measure for that class (whose t^{th} element is denoted with $g(t)$, and whose explicit formula is based on pool accuracies estimated through validation data [23]) to get $\mu_i(\boldsymbol{d}_i(\boldsymbol{f}_T))$. Therefore, for every test instance $\boldsymbol{f}_T$, L vectors of length K are evaluated, each corresponding to a class and containing values of the considered fuzzy measure.

   Furthermore, we will consider the following *trainable linear combinations*. The first is based on the `K weights` approach, defined as $\mu_i(\boldsymbol{d}_i(\boldsymbol{f}_T)) \triangleq \widetilde{\boldsymbol{w}}^T \boldsymbol{d}_i(\boldsymbol{f}_T)$, where $\widetilde{\boldsymbol{w}} \in [0,1]^{K\times 1}$ and $\widetilde{w}_k \triangleq \frac{(1/\epsilon_k)}{\sum_{t=1}^{K}(1/\epsilon_t)}$, being ϵ_k the (estimated) error-rate of k^{th} classifier [40]. Secondly, the `KL weights` approach is based on [41] $\mu_i(\boldsymbol{d}_i(\boldsymbol{f}_T)) \triangleq \boldsymbol{w}_i^T \boldsymbol{d}_i(\boldsymbol{f}_T)$, where $\boldsymbol{w}_i \triangleq (\boldsymbol{D}_i\boldsymbol{D}_i^T)^{-1}\boldsymbol{D}_i\boldsymbol{b}_i$ and $\boldsymbol{D}_i \in [0,1]^{K\times N}$ denotes the matrix obtained arranging all the i^{th} columns of the DP matrices belonging to the validation set, whereas $\boldsymbol{b}_i \in \{0,1\}^N$ whose n^{th} entry equals 1 when the corresponding sample of the validation set belongs to c_i.

3. *(CI) Decision Templates (DT)*: the DT approach [23] stores the *most typical* DP for each class c_i (i.e. the DT of i^{th} class, denoted with $\bar{\boldsymbol{D}}_i$) and then compares it with the current DP matrix $\boldsymbol{D}(\boldsymbol{f}_T)$ using a suitably chosen similarity measure $\mathcal{S}(\boldsymbol{D}(\boldsymbol{f}_T), \bar{\boldsymbol{D}}_i)$). The confidence for i^{th} class will be then

$$\mu_i(\boldsymbol{D}(\boldsymbol{f}_T)) \triangleq \mathcal{S}(\boldsymbol{D}(\boldsymbol{f}_T), \bar{\boldsymbol{D}}_i)$$

   when a new test instance $\boldsymbol{f}_T$ is submitted. Differently, during the training phase, the DT associated to i^{th} class $\bar{\boldsymbol{D}}_i$ is built as the average of the all the DP matrices within the validation set labelled with c_i.

   In this study, we will employ three common similarity measures for the DT testing phase, based on the following distances [23]: (a) squared Euclidean (`DT-SE`); (b) ℓ_1 norm after vectorization (`DT-L1`); (c) symmetric fuzzy-set originated (`DT-FSD`).

4. *(CI) Dempster-Shafer approach (DS)*: the present combiner takes its inspiration from the theory of evidence (DS theory). Similarly to DT method, in DS approach the DT matrices $\bar{\boldsymbol{D}}_1, \ldots, \bar{\boldsymbol{D}}_L$ are evaluated from the validation set. On the other hand, the similarity evaluation between each $\bar{\boldsymbol{D}}_i$ and the DP matrix $\boldsymbol{D}(\boldsymbol{f}_T)$ is replaced by the following steps [23].

   First, a $L \times K$ *proximity matrix* $\boldsymbol{\Phi}$ is built, whose $(i,k)^{th}$ entry is a normalized measure of distance[10] between the k^{th} rows of the i^{th} class DT $\bar{\boldsymbol{D}}_i$ and of the DP (a similarity measure between the confidence vector of k^{th} classifier and its *typical profile* when c_i is the actual class). Secondly, by using $\boldsymbol{\Phi}$, for every class $c_i \in \Omega$ and for every classifier $k = 1,\ldots,K$, a *belief* degree $\beta_i(\boldsymbol{r}_k(\boldsymbol{f}_T))$ is computed. Finally, the i^{th} degree of support $\mu_i(\boldsymbol{D}(\boldsymbol{f}_T))$ is obtained as a normalized product of the belief degrees $\beta_i(\boldsymbol{r}_k(\boldsymbol{f}_T))$, $k = 1,\ldots,K$.

6. Experimental Results

In this section we first provide a detailed description of the dataset used (Sec. 6.1). Then, in Sec. 6.2, we recall the performance metrics evaluated in our analysis. We then report a systematic investigation of the effectiveness of different pre-processing operations performed on data before actual classification (Sec. 6.3, so as to underline "best practices") by measuring their influence on the performance of all the considered classifiers/combiners. Finally, in Sec. 6.4 we report performance of the proposed MCS (and investigate its modularity) in comparison to state-of-the-art classifiers devised for mobile TC.

[9] In this paper we have set $\alpha = \frac{1}{2}$ and $P\% = 20\%$ for `Generalized` and `Trimmed Mean` combiners, respectively.

[10] Although any distance could be employed, in this study we concentrate on ℓ_2 norm (`DS-L2`) for simplicity.

6.1. Dataset Description

The considered dataset is composed of real-traffic traces, provided by an international mobile solutions provider (already anonymized) and generated from a total of 49 apps (resp. 45) on Android (resp. iOS) devices, run *separately*. Mobile traffic has been generated by different *human-users* running various devices, without specific constraints on the operating system / app version, being the latter a *worst case* for mobile TC [30]. As specified in Sec. 1, we are not allowed (due to NDA) to provide details on the network where the traffic traces were collected. Accordingly, ground truth is obtained by labeling (manually) each trace with the generating application. As a whole, the dataset is made up of 607 (resp. 419) traffic traces, with an average duration of 282 (resp. 296) seconds and $1 \div 60$ (resp. $1 \div 48$) traces per app.

The traces belonging (viz. the dataset corresponding) to the two aforementioned operating systems are investigated separately, in order to evaluate the detectability of mobile apps in a well established scenario (i.e. belonging to the same operating system / store).[11]

After the burstification process described in Sec. 3.1, network traffic is then processed using the (statistical features extraction) approach introduced in Sec. 3.2. We remark that the minimum SB length considered in this study is 7 (as suggested in [10]), since it is the shortest sequence of packets representing a meaningful data transfer which includes a TCP handshake and an HTTP request/response with corresponding ACKs. On the other hand, in this work, we do not restrict superiorly the length of the SB to be analyzed, since we did not consider (for reasons of computational complexity) classification algorithms taking as input the (*varying*) raw vector of packets (referred to as "per-flow length classifiers" in [10]). We observe that, given the collection methodology of the considered traces, the SB definition is not prone to possible wrong-segmentation of the SBs within the same burst, according to the aggregation principle of the same destination IP address / port couple.

Additionally, the number of instances for each app presents a severe imbalance (this is especially true for the least observed ones). For an excellent introduction to different techniques which can be applied to imbalanced datasets, with specific focus on Internet TC, please refer to [42]. As a summary, two different "philosophies" may be pursued for dealing with class imbalance. These mainly pertain to re-sampling (comprising oversampling and undersampling) methods [42] and cost-sensitive learning [12, 43] approaches.

In this paper, an oversampling procedure has been applied to the dataset. More specifically, we applied the *Synthetic Minority Oversampling TEchnique* (SMOTE) [44] to the apps with a number of SBs less than 30^{th} percentile of the distribution of the number of SBs per app[12], in order to obtain a reasonable number of samples per app. SMOTE is one of the most popular approaches for data-based class-minority oversampling. Specifically, we adopted the filter implemented in the Weka environment by means of `weka.filters.supervised.instance.SMOTE` Java class. We remark also that the results obtained with different percentages of SMOTE (e.g., corresponding to the 40^{th} and 50^{th} percentiles) have shown no discrepant relative performance among the classifiers and combiners (both hard and soft) considered in what follows, thus underlining the *stability* of the considered dataset.[13]

6.2. Performance Metrics

The successive analysis will be based on the following performance metrics [23]: (*i*) *overall accuracy*, (*ii*) *precision*, and (*iii*) *recall* [45]. Since the latter two metrics are defined on a per-app (per-class) basis, we will employ their arithmetically averaged (viz. macro) versions. Additionally, we will consider (*iv*) the *F-measure*, defined as ($\mathrm{F} \triangleq (2 \cdot \mathrm{prec} \cdot \mathrm{rec})/(\mathrm{prec} + \mathrm{rec})$), being a scaled harmonic mean of (macro-) precision and (macro-) recall, so as to account for both the effects of precision (`prec`) and recall (`rec`) in a concise fashion. Besides, we will consider (*v*) *confusion matrices* of classifiers (resp. combiners) to provide their whole performance "picture" and identify the most frequent misclassification patterns. Clearly, a higher concentration toward the diagonal (where predicted app equals the actual one) implies better performance of the generic classifier (resp. combiner).

Finally, we remark that each considered setup will employ a random training-validation-test set splitting (with corresponding percentages $50\% - 25\% - 25\%$, respectively).

6.3. Burst Threshold Impact on Performance and Hints on Dataset Pre-processing

Our first investigation on pre-processing steps applied to considered traces was aimed at assessing whether there is a substantial gain (or, generically, a significant change) in performance when cleaning traffic traces from *TCP retransmissions*. Results (not shown here for the sake of brevity) have underlined (almost) insensitivity of performance to the aforementioned operation, quantified in less than 0.2% change in accuracy for the best base classifier observed when considering a SB definition corresponding to $1s$ of BT. For this reason, in what follows, we have processed uncleaned (i.e. including TCP retransmissions) traffic traces, as the above step does not affect classification performance in a substantial way while adding unnecessary complexity to the proposed classification approach.

Then, two (coupled) useful investigations are pursued in what follows. First, we analyze the sensitivity of the classification performance to SB definition, focusing on the BT, to analyze whether (and, in the affirmative case, to which degree) classifiers' performance are affected by this parameter. Indeed,

[11] Nevertheless, since common apps between the two datasets are 42 (i.e. 85.7% and 91.3% of the whole Android and iOS dataset, respectively), an OS-agnostic classification concerning the possibility to distinguish not only the specific app, but also the OS it belongs, represents an interesting further avenue.

[12] Note that this distribution depends on the initial number of SBs (without SMOTE) and therefore on the value of the BT.

[13] We remark that the present framework does not necessarily rely on SMOTE and such procedure can be safely removed from the pipeline in the case of a larger dataset.

previous studies have only provided results pertaining to empirically chosen values of the BT, corresponding to $1s$ [10, 29] and $4.5s$ [27], respectively. Hence, to provide a comprehensive BT analysis, we have employed the interval $[0.5, 5]$ seconds (with increments of $\Delta = 0.5s$) which includes *both* the aforementioned *empirical choices*.

Secondly, the aim is to investigate the potential gain achievable when removing zero-payload traffic. Indeed, a similar preprocessing step has been suggested in [46] for a *website fingerprinting task*. Specifically, it has been advocated to remove packets sized 52 from features' evaluation, based on the intuition (confirmed by the appealing results in Panchenko et al. [46]) that packets of this length occur for all possible web pages, as these correspond to acknowledgments between sender and receiver (TCP ACK packets with no payload). Thus, an evaluation of features without packets sized 52 would allow to discard non-website-specific behavior (which may be regarded as *noise* for the evaluation of the considered features). We argue that this may be also the case of mobile TC, as these packets correspond to a non-app-specific behavior. Accordingly, we pursue a similar (though more general, as some packets sized 52 could correspond to mobile data traffic exchange) approach, by *removing packets with zero-payload*.

Since the two aforementioned processing steps are *interdependent* (i.e. the BT is influenced by the presence/absence of zero-payload packets), the following three configurations of the dataset have been considered, by varying BT value:

- (a) Original dataset (no pre-processing);
- (b) Dataset with zero-payload packets removed *after* the SB extraction;
- (c) Dataset with zero-payload packets removed *before* the SB extraction.

Finally, SBs with a length less than *"Min Length"* packets (see Fig. 1) have been discarded in order to improve classification performance, as suggested in [10]. Note that in [10] a (minimum) flow length of 7 packets is considered as the optimal choice, since it represents the length of the shortest "complete" SB, that consists of a TCP handshake (three packets), an HTTP request/response pair (two packets) and the corresponding acknowledgments (two packets). In this work, we have made the same choice for *both* the cases (a) and (b). On the other hand, in case (c), we have selected a Min Length equal to 2, since in the latter case SB extraction is performed on traffic traces whose zero-payload packets have been already removed (i.e. TCP handshake and acknowledgments belonging to the shortest "complete" SB).

Fig. 2 shows the number of SBs in these three datasets as a function of the BT. Intuitively, the greater the BT, the lower the number of (resp. the longer the) SBs obtained (for all the datasets). In detail, this number ranges from 16939 (resp. 15166) to 43089 (resp. 35064) for the dataset with zero-payload packets removal after (with a $5s$ BT) and before (with a $0.5s$ BT) the SB extraction, respectively. From inspection of the figure, it can be noticed a "slope" change at BT equal to $1s$. It can be inferred that, when BT is less than $1s$, the burstification process leads to an excessive fragmentation, and does not adequately capture the bursty nature of the considered mobile traffic. On the other hand, values higher than $1s$ may represent solutions which may lead to *merging* actually-distinct SBs (although in the range $(1, 5]$ seconds such "merging effect" seems not dramatic).

In Fig. 3 both the accuracy (top) and the F-measure (bottom) for all the classifiers described in Sec. 4 are shown vs. the BT value, for Android traces. More specifically, for each figure, cases (a) full dataset, (b) zero-payload packets removed after SB extraction, and (c) zero-payload packets removed before SB extraction are reported in left, middle and right boxes, respectively.

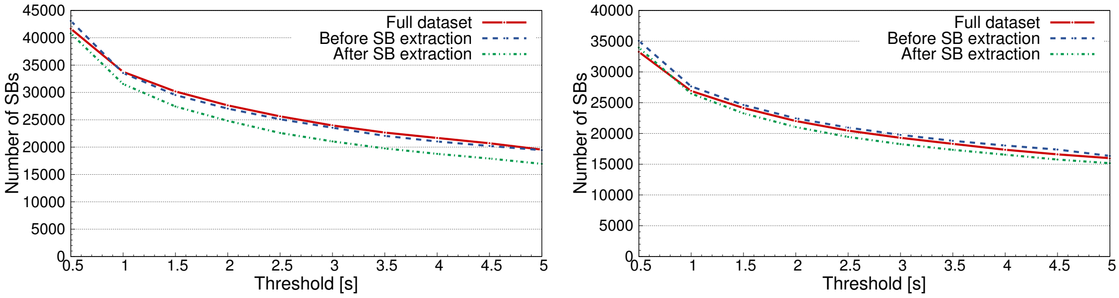
From the inspection of results, the highest performance is obtained with a threshold of $1s/1.5s$ in the case (c), i.e. with zero-payload packets removed before SB extraction. The optimal BT value found numerically also confirms the considerations on fragmentation/merging traffic effects arising from an inaccurate (viz. lower/higher) choice of the BT value, somewhat anticipated by the slope change phenomenon in Fig. 2. This trend can be observed for both Android and iOS traces (not shown for brevity). The results agree qualitatively with the considerations in [10, 29], thus underlining that $1s$ represents a good and stable choice for the BT.[14]

Similarly, it is evident that removal of zero-payload packets *always* provides some gain in performance, which is independent on the specific BT considered, and whether such removal is performed after (b) or before (c) SB extraction. Nevertheless, an additional performance improvement is obtained when performing the removal before SB extraction (c). This may be explained as this filtering of "noisy packets" is also beneficial for a more effective SB segmentation. Indeed, taking into account a threshold value of $1s$, for the best base classifier (i.e. `Tay_RF`), removal of zero-payload packets before and after SB partitioning produces an accuracy increment of +6.7% (resp. +9.7%) and +3.3% (resp. +4.3%), respectively. Thus, as stated above, when this zero-payload cleansing is performed before, a further enhancement of +3.4% (resp. +5.4%) can be obtained.

Interestingly, F-measure increments (being in this case `Her_Cos` the best base classifier in terms of F-measure) are markedly smaller and substantial only for the removal of zero-payload packets before SB extraction: +1.5% (resp. +3.6%).

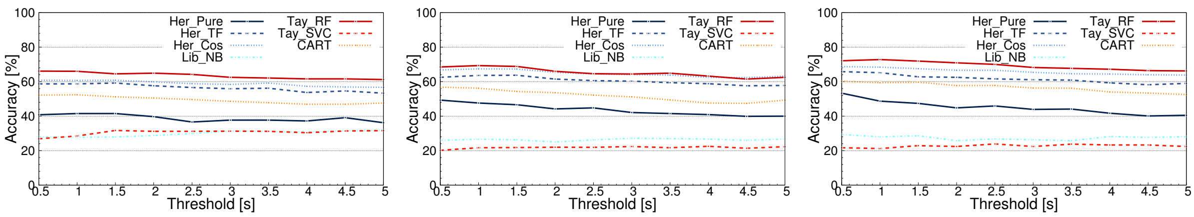
Additionally, it is apparent a weakly-decreasing trend for best performing classifiers (namely `Tay_RF`, `Her_Cos`, and `Her_TF`) for increasing values of the BT. Similar trends can be observed for considered hard and soft combiners although less evident, since the influence of other base classifiers. The aforementioned behavior can be explained as larger values of the BT imply *longer* SBs, thus precluding a correct segmentation of the different actions associated to a certain app during time.

[14]Nonetheless, in any case, automatic design (and adaptability) of this value would be desirable, being able to cope with networks experiencing different delay conditions.

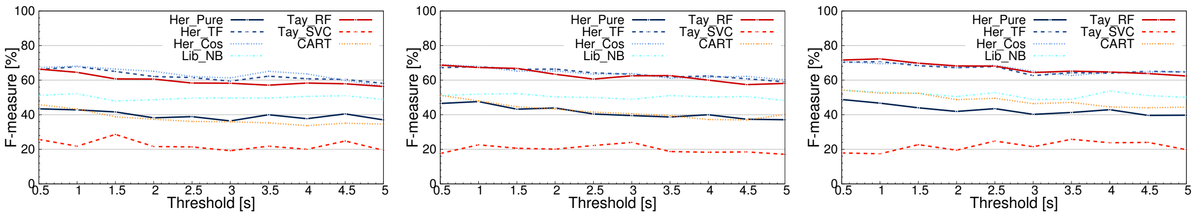


(a) Android dataset. (b) iOS dataset.

Figure 2: Number of service bursts for different values of the BT considering Android (a) and iOS (b) datasets.



(a) Full dataset. (b) Zero-payload packets removed *after* SB extraction. (c) Zero-payload packets removed *before* SB extraction.



(d) Full dataset. (e) Zero-payload packets removed *after* SB extraction. (f) Zero-payload packets removed *before* SB extraction.

Figure 3: Accuracy (a-c) and F-measure (d-f) of the base (state-of-the-art) classifiers to varying the BT (Android dataset).

Therefore, since the removal of zero-payload packets before SB extraction seems an appealing pre-processing step over a wide range of BT values, in what follows we compare the performance of (*a*) the best base classifier (corresponding to `Tay_RF`, thus qualitatively agreeing with the results in [10]), (*b*) the best hard combiner (corresponding to either `NB` or `WMV` combiner, depending on the specific performance metric deemed relevant) and (*c*) the best soft combiner (corresponding to the `KL weights`).

The present investigation is conducted by measuring their accuracy and F-measure as a function of the BT (over the same threshold range employed for Fig. 3), in Fig. 4 for Android traces (similar results have been observed iOS traces).

This allows investigating the general improvement provided by the present MCS system over the best base classifier, either considering hard or soft techniques, which is seen to be *almost independent* on the specific burst threshold considered. For completeness, accuracy plots in Fig. 4a also report the performance of `ORA` combiner.[15], which highlights how the proposed approach "pushes" the performance toward the combining theoretical performance (i.e. upper-bound) for the considered pool.

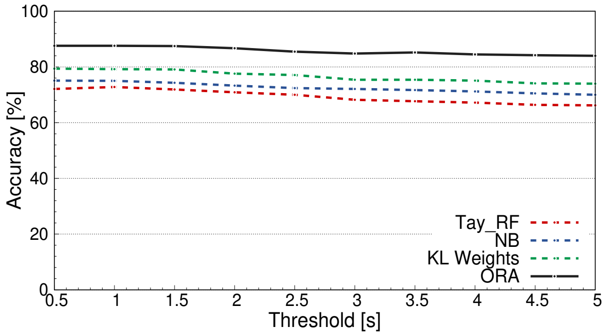
6.4. Classification Results

Based on the previous considerations, in what follows we focus on case (*c*) (that is, removing zero-payload packets before burstification) and set the BT to $1s$, collectively representing the scenario with the highest performance observed. Then, we show results (at a finer detail) obtained by the application of the proposed MCS (see Sec. 3) to the aforementioned case.

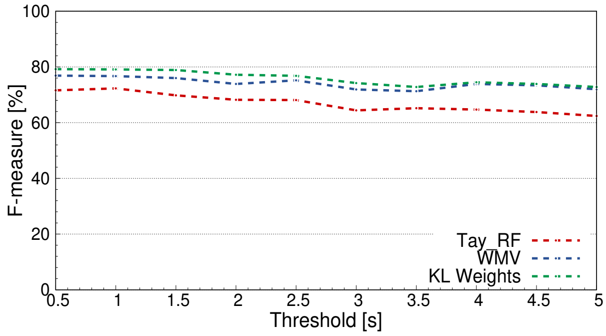
First, in Tab. 2, we report the performance of all the base classifiers described in Sec. 4, in terms of the considered synthetic measures. Also, for completeness, we report the accuracy and recall achieved by the `ORA` (rightmost column).

From inspection of results, it is apparent that `Tay_RF`, `Her_Cos`, and `Her_TF` achieve the highest performance w.r.t.

[15] Indeed, precision (and consequently F-measure) of the `ORA` cannot be evaluated since its error patterns are not defined [19].



(a) Accuracy.



(b) F-measure.

Figure 4: Accuracy (a) and F-measure (b) of the best base classifier, hard and soft combiner versus the BT (Android dataset). Performance refers to the dataset with zero-payload packets removed *before* the SB extraction.

the considered measures in the present setup, being still prone to classification errors. The quantitative scores are (only at first glance) in contrast to those typically observed in Internet TC [19] and, more recently, to those achieved in the mobile context [10, 29]. However, in the former case, the classification problem is simplified by a homogeneous and less dynamic nature of the traffic being observed (while typically coping with a lower number of classes to discriminate from), whereas in the latter case it likely pertains to a non-exhaustive traffic collection procedure, being bot-generated and probably not capable of adequately "representing" all the "paths" of a generic app. Additionally, by looking at `ORA` performance, the best accuracy (resp. recall) of the base classifiers can be improved by means of the proposed MCS up to 14.8% (resp. 19.5%) for Android and up to 16.8% (resp. 19.9%) for iOS, respectively. We notice that the upper-bound performance may be further improved by the adoption in the pool of other classifiers suitably-devised for mobile TC, underlining *the appeal of the MCS*.

To this end, in Tab. 3 we show (and compare) the performance of the considered hard combiners. Results underline that `BKS` is able to provide the highest improvement with respect to the best base classifier (`Tay_RF`) in terms of overall accuracy. The same reasoning applies to `NB` for recall measure (the latter also performs quite well in terms of accuracy). Differently, `MV` and `WMV` result appealing because of the remarkable improvement in terms of precision and F-measure over the best base classifier (between +3.8% and +5.4%, respectively).

Interestingly, `WMV` and `NB` represent the most appealing choices in terms of the set of performance metrics considered, (almost) collectively providing the highest performance. This is explained as they are less prone to over-fitting (and have less training requirements), while also enjoying lower complexity w.r.t. `WER` and `BKS`. Remarkably, all the considered hard combiners (except for `MV`, when referring to the sole overall accuracy experienced with Android traffic), outperform the best base classifier in terms of all the considered performance metrics. This holds in both iOS and Android traffic.

A similar numerical comparison is shown in Tab. 4, where the performance of the three different groups of soft-combiners considered in this study are reported in separate sub-tables (i.e. CC non-trainable, CC trainable, and CI in sub-tables (*a*-*b*), (*c*), and (*d*), respectively), so as to (possibly) underline an interesting performance trend of a given group. For each group, `ORA` performance (as the rightmost column) is reported so as to highlight the corresponding improvement achievable.

By looking at their performance, it is apparent that a remarkable performance improvement can be achieved already with the sole use of CC non-trainable combiners (for which the availability of validation data is *not needed*). In fact, for the considered case, the `Mean`, the `Median`, the `Trimmed Mean` and the `Generalized Mean` are able to improve `Tay_RF` performance in terms of all the reported synthetic measures. This holds in both iOS and Android traffic. On the other hand, the soft combination approaches provided by `PP`, `Maximum`, `Minimum`, `Harmonic Mean`, and `Geometric Mean` always lead to *unsatisfactory* performance when compared to the best base classifier (`Tay_RF`). This can be explained as these are more sensitive to soft-output misspecification of the classifiers in the pool.

Interestingly, a general (remarkable) improvement is achieved by the *whole group* of CC trainable combiners over `Tay_RF`. More specifically, though all the combiners within this group perform quite well, `KL weights` represents the most appealing choice in terms of all the measures considered (and for traffic belonging to different OSs).

Furthermore, CI combiners are also able to improve, in most cases (except for a slight degradation of precision measure, see later discussion), performance over the best base classifier, with `DT-L1` and `DS-L2` performing slightly better than others in the CC group. Still, the larger generalization capability of CI combiners does not pay back in terms of performance in comparison to CC trainable combiners. This may be attributed to an inadequate number of validation samples or to an over-fitting phenomenon. From a direct comparison of all the combiners belonging to all groups reported (both hard and soft), it is evident that `KL weights` represents the *best combiner* considered in this study for the present dataset. Finally, we underline that improved absolute performance measures may be achieved by the proposed MCS if additional (high performing and/or diverse) classifiers are developed to enlarge the considered pool.

Additionally, to summarize the improvement achieved by

Table 2: Performance (%) of base (state-of-the-art) classifiers considering Android (iOS) traffic.

Classifier	Her_Pure	Her_TF	Her_Cos	Lib_NB	Tay_RF	Tay_SVC	CART	ORA
Accuracy	48.7 (50.9)	65.2 (64.8)	68.4 (68.9)	28.0 (32.4)	**72.8 (70.9)**	21.2 (27.4)	59.4 (56.7)	87.6 (87.7)
Macro Precision	45.1 (47.2)	74.6 (70.0)	71.2 (69.3)	60.3 (60.7)	**74.7 (71.5)**	21.4 (30.0)	52.8 (50.9)	-
Macro Recall	54.8 (49.9)	58.4 (56.8)	63.5 **(62.3)**	36.0 (33.6)	**64.1 (62.3)**	9.89 (14.2)	51.4 (49.3)	83.6 (82.2)
Macro F-Measure	46.7 (47.7)	70.7 (66.9)	69.5 (67.8)	53.1 (52.3)	**72.3 (69.4)**	17.4 (24.6)	52.5 (50.6)	-

Table 3: Performance (%) of hard combiners considering Android (iOS) traffic.

Combiner	MV	WMV	REC	NB	BKS	WER	ORA
Accuracy	72.2 (71.9)	72.8 (72.4)	73.8 (72.6)	**75.0** (74.0)	**75.0 (74.3)**	73.8 (71.9)	87.6 (87.7)
Macro Precision	79.3 **(76.9)**	**80.1 (76.9)**	78.7 (76.3)	75.8 (73.5)	77.4 (74.2)	75.6 (72.1)	-
Macro Recall	65.4 (63.4)	65.8 (63.9)	67.0 (64.2)	**70.7 (67.6)**	69.7 (67.2)	65.7 (63.5)	83.6 (82.2)
Macro F-Measure	76.1 (73.8)	**76.7 (73.9)**	76.1 (73.6)	74.7 (72.3)	75.7 (72.7)	73.4 (70.2)	-

each group of (hard/soft) combining techniques, we have reported in Tab. 5 the *Maximum Improvement Over the Best Classifier* (MIOBC) provided by each group for every performance measure. Referring to the group of hard combiners, it is apparent that such group is *always* able to provide an improvement, ranging from +2.2% (accuracy on Android traffic) to +6.6% (recall on Android traffic), by means of diversity principle, representing the milestone for adoption of classifier fusion techniques. However, as remarked before, different approaches (namely MV, WMV, NB, and BKS) result best according to different performance metrics. A similar reasoning applies to the group of CC non-trainable combiners (second column), where the improvement ranges from +3.0% (accuracy on Android traffic) to +6.5% (recall on Android traffic). Here, the improvement is qualitatively similar to hard combiners. However, CC non-trainable combiners, though requiring the availability of soft outputs from each classifier in the pool, do not require the availability of a validation set (which is instead required in the design of almost all the hard combiners here considered). This may be appealing in the case of scarcity of additional training (validation) data. On the other hand, the group of CC trainable combiners is able to provide the best improvement for each metric. This is not only the consequence of the presence of KL weights within the group, having the highest performance. Indeed, as observed earlier, all the combiners within the group are able to provide significant gains. Finally, CI combiners are able to collectively provide a performance improvement over almost all the considered metrics. The sole exception is represented by precision (which is slightly degraded for all the combiners within this group), with a consequent gain reduction of the corresponding highest F-measure achieved by the CI group.

We now compare the performance of the best classifier with the best hard and soft combiners at a *finer detail*, that is, by analyzing their confusion matrices, which allow to focus on misclassification patterns among apps (cf. Sec. 6.2). To this end, in Figs. 5a, 5b and 5c, we show the confusion matrices of Tay_RF, NB and KL weights, respectively. The reported matrices refer to Android traffic. However it is worth noticing that similar qualitative trends have been observed for iOS traffic. From inspection of the results, it is revealed a homogeneously-reduced occurrence of misclassification patterns when employing a (good) combiner with respect to the best base classifier. This is more evident when a soft combiner is employed.

Finally, in Tabs. 6 and 7, we delve into how classifiers subset selection affects performance, focusing on the F-measure. The intent is investigating possible performance gain of the considered combiners (grouped as done previously) and computational complexity reduction, by discarding *non-informative* classifiers from the pool. Since the number of different subsets is combinatorial and having available different optimization criteria (combiners), it is impractical evaluating performance for all the possible combinations. Hence, we adopt an heuristic approach informed by the *diversity* of classification methods and iteratively removing the *worst performing* classifier.

Referring to hard combiners (cf. Tab. 6), several observations can be made. The best overall F-measure performance is achieved by MV and REC on iOS and Android traffic, respectively. The appeal of this result is that these combiners have low requirements both in terms of training samples and operational (testing phase) complexity. Additionally, it is apparent that the hard combiners requiring the least parameters to be trained (i.e. MV, WMV, REC, and NB) *all benefit* from the selection of a subset of classifiers within the pool. Interestingly, they all achieve their *maximum per combiner* when only Her_Cos, Lib_NB, and Tay_RF are employed. This may be attributed at the higher diversity provided by these three different base classifiers. On the other hand, WER also presents improved performance with a different selection of the subset of classifiers (namely, a larger subset for Android traffic, whereas only Her_Cos and Tay_RF are needed in the pool to achieve its highest performance over iOS traffic). Finally, it is apparent that *BKS does not benefit from the same subset selection as MV, WMV, REC, and NB.* Therefore, we argue that this may be attributed to over-fitting issues (i.e. unnecessarily modeled correlation between diverse base classifiers).

Then, with reference to CC non-trainable combiners (cf. Tabs. 7a and 7b), we first observe that PP, Maximum, Minimum, Harmonic Mean, and Geometric Mean combiners have a *dramatic improvement* of F-measure performance when considering small subsets of the classifiers pool. Similarly, the Mean, Median, Trimmed Mean, and Generalized Mean are able to improve (almost always) their performance when considering the smallest pool composed by Her_Cos and Tay_RF. However, their performance improvement is less steep. This trend may be

Table 4: Performance (%) of different classes of soft combiners, considering Android (iOS) traffic.

(a) Class-conscious (CC) non-trainable combiners (1).

Combiner	`Mean`	`Maximum`	`Minimum`	`Median`	`PP`	`ORA`
Accuracy	75.3 (73.8)	61.5 (56.7)	51.8 (47.3)	73.7 (72.8)	52.1 (47.6)	87.7 (87.6)
Macro Precision	74.7 (71.8)	55.2 (50.2)	38.1 (35.3)	**79.8 (77.4)**	38.4 (35.6)	-
Macro Recall	**70.6** (67.5)	54.5 (50.5)	44.4 (40)	67.1 (64.2)	44.7 (40.3)	83.7 (82.3)
Macro F-Measure	73.8 (70.9)	55 (50.3)	39.3 (36.1)	**76.9 (74.3)**	39.5 (36.5)	-

(b) Class-conscious (CC) non-trainable combiners (2).

Combiner	`Trim Mean`	`Harm Mean`	`Geom Mean`	`Gen Mean`	`ORA`
Accuracy	75.6 **(74.4)**	51.8 (47.1)	52.3 (47.9)	**75.8 (74.4)**	87.7 (87.6)
Macro Precision	77.7 (75.1)	38.2 (35.1)	38.5 (35.6)	77.1 (74.9)	-
Macro Recall	70.4 **(67.7)**	44.4 (39.9)	44.5 (40.3)	70.5 (67.3)	83.7 (82.3)
Macro F-Measure	76.2 (73.5)	39.3 (35.9)	39.6 (36.5)	75.7 (73.2)	-

(c) Class-conscious (CC) trainable combiners.

Combiner	`Fuzzy Integral`	`K weights`	`KL weights`	`ORA`
Accuracy	75.4 (73.5)	76.1 (74.2)	**79.2 (77.8)**	87.7 (87.6)
Macro Precision	77.0 (73.6)	76.3 (72.7)	**80.6 (78.2)**	-
Macro Recall	70.4 (67.2)	71.1 (67.8)	**73.6 (71.6)**	83.7 (82.3)
Macro F-Measure	75.6 (72.2)	75.2 (71.7)	**79.1 (76.8)**	-

(d) Class-indifferent (CI) combiners:
Decision Templates (DT) and Dempster-Shafer (DS) approaches.

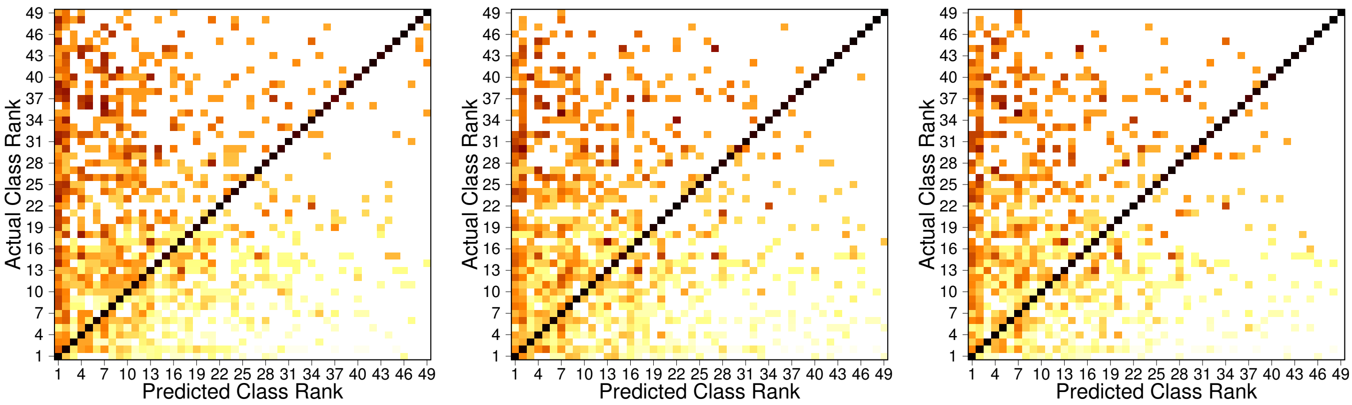
Combiner	`DT-SE`	`DT-L1`	`DT-FSD`	`DS-L2`	`ORA`
Accuracy	75.6 (72.6)	74.9 **(73.8)**	74.7 (72.8)	**75.7** (73.0)	87.7 (87.6)
Macro Precision	73.3 (69.1)	73.2 **(71.4)**	73.2 (69.9)	**73.5** (69.9)	-
Macro Recall	**72.6 (69.1)**	71.1 (68.4)	69.9 (66.8)	72.2 (69.0)	83.7 (82.3)
Macro F-Measure	73.1 (69.1)	72.8 **(70.7)**	72.5 (69.3)	**73.2** (69.7)	-

Table 5: *Maximum Improvement Over Best Classifier* (MIOBC) of the F-measure (%) for each class of hard and soft combiners, considering Android (iOS) traffic.

Combiner	*Hard*	*CC non-trainable*	*CC trainable*	*CI*
Accuracy	+2.2 (+3.4)	+3.0 (+3.5)	**+6.4 (+6.9)**	+2.8 (+2.9)
Macro Precision	+5.4 (+5.4)	+5.1 (+5.9)	**+5.9 (+6.7)**	*-1.4 (-0,1)*
Macro Recall	+6.6 (+5.3)	+6.5 (+5.4)	**+9.5 (+9.3)**	+8.5 (+6.8)
Macro F-Measure	+4.4 (+4.5)	+4.6 (+4.9)	**+6.8 (+7.4)**	+0.8 (+1.3)

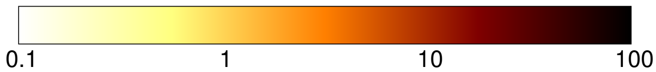
explained as CC non-trainable combiners are more prone to be biased from wrong classifiers in the pool, due to the lack of high-level (validation-based) training. Nevertheless, the latter sub-group possesses an intrinsic robustness (due to their peculiar combination functions) to having *outliers* in the pool. On the other hand, by observing performance of CC trainable combiners (cf. Tab. 7c), it is apparent that `KL weights` benefits from a judicious use of the subset. This allows reducing the number of parameters to be trained, especially those related to weak classifiers, and thus avoid slight over-fitting. A different trend is instead observed for `K weights`, being similar to that observed for CC non-trainable combiners. The reason is that the linear (separating) vector employed is based on the assumption that each soft-output well-matches (i.e. except for some estimation noise) the actual one [40]. Therefore, this approach is potentially sensitive to erroneous (i.e. providing incoherent soft-outputs) base classifiers. A somewhat similar behavior as `BKS` is observed for `Fuzzy Integral`, which *does not benefit* from subset selection. This may be attributed to the fact that the proposed fuzzy-based fusion design is resistant to classifiers' uncertainty. A less evident trend can be drawn for CI combiners (cf. Tab. 7d). Nonetheless, it can be concluded how all the proposed approaches achieve the highest F-measure with the group considered by `Her_Cos`, `Lib_NB`, and `Tay_RF`, with the sole exception of `DT-FSD` in Android traffic, where the smallest group composed by `Her_Cos` and `Tay_RF` should be employed to reach the highest performance.

A summarizing comparison, reporting the MIOBC (in terms of F-measure) for each group of combiners, is shown in Tab. 8, exploring the same subset choices previously considered. From its inspection, it is apparent how improved performance (with respect to considering the whole pool of classifiers) can be obtained on Android traffic (+0.5%) or the same results with less training requirements (i.e. a CC non-trainable combiner) in iOS traffic.



(a) Best base classifier (`Tay_RF`). (b) Best hard combiner (`NB`). (c) Best soft combiner (`KL weights`).

1 = QQ	2 = SayHi	3 = GooglePlay	4 = eBay	5 = HotSpot
6 = 6Rooms	7 = PureVPN	8 = QQReader	9 = PaltalkScene	10 = HidemanVPN
11 = Anghami	12 = BaiDu	13 = Google+	14 = Hooq	15 = Repubblica
16 = 80sMovie	17 = Narutom	18 = IFengNews	19 = GoogleMap	20 = PrivateTunnelVPN
21 = GoogleAllo	22 = MeinO2	23 = QianXunYingShi	24 = Palringo	25 = Hangouts
26 = GroupMe	27 = Crackle	28 = LRR	29 = InterVoip	30 = NetTalk
31 = EFood	32 = RaidCall	33 = Go90	34 = FSecureVPN	35 = NileFM
36 = Shadowsocks	37 = Ryanair	38 = Sogou	39 = Guvera	40 = SmartVoip
41 = 9YinZhenJing	42 = Minecraft	43 = 360Security	44 = GooglePhotos	45 = GoogleCast
46 = Hidemyass	47 = FrostWire	48 = Mobily	49 = RiyadBank	



(d) Log-scale (%) and labels.

Figure 5: Confusion matrices of the best (a) base classifier, (b) hard combiner, (c) soft combiner (Android dataset). Note that the labels (d) are ranked according to decreasing abundance of samples and the logarithmic scale (d) is used to evidence small errors.

7. Conclusions and Future Directions

We tackled TC of mobile apps by proposing a MCS encompassing the following classifier fusion techniques: hard combiners (based on Type I classifiers) and soft combiners (based on Type III classifiers) [23]. For the second fusion approach, several soft-combination approaches belonging to three different philosophies have been explored: (*a*) CC non-trainable, (*b*) CC trainable, and (*c*) CI combiners. The considered MCS has been employed with a pool of 7 state-of-the-art classifiers specific or suitable for mobile traffic [7, 8, 10]. Its evaluation has been performed on an actual dataset describing traffic from 49 (resp. 45) apps in Android (resp. iOS) devices provided by a solution provider.

The results have shown a performance gain of the MCS over the best base classifier up to 9.5% (referring to macro recall in the case of Android traffic). Such improvement has been also shown to be quite general over *different apps* considered, given the homogeneously-reduced error-patterns observed by comparing the confusion matrices of the best base classifier and best (soft/hard) combiner. Nonetheless, the modularity of the considered MCS allows its virtual application to other suitably-devised classifiers for further performance enhancement.

The proposed framework has been also used to validate the design of a novel pre-processing procedure for traces before feature evaluation, highlighting that removing zero-payload packets before temporal segmentation of traces into SBs resulted in the *highest performance*. Conversely, traces cleansing from TCP retransmissions has been found to be irrelevant in terms of performance.

Finally, a further investigation of MCS performance versus subset selection of base classifiers (as well as `ORA` results) has highlighted further improvement toward optimal (and low-complexity, as the same performance of the whole classifiers set has been obtained with a very small pool of selected classifiers) combination of base classifiers.

Future directions will include: (*i*) a deeper analysis with an enlarged pool, made of classifiers (possibly) fed with a specifically-optimized set of features (selected by means of information-theoretic measures and whose stability, with respect to the dynamic nature of moving traffic, needs to be carefully evaluated [17, 18]), (*ii*) an intelligent pool subset selection, (*iii*) the evaluation of the sampling impact [47], (*iv*) the analysis of the proposed MCS in an early-TC context, (*v*) the development of a MCS able to directly deal with imbalanced training data through cost-sensitive learning of classifiers and combiners [42, 43], and (*vi*) the implementation of the classifiers and combination techniques in TIE [20].

References

[1] S. Valenti, D. Rossi, A. Dainotti, A. Pescapè, A. Finamore, M. Mellia, Reviewing traffic classification, in: Data Traffic Monitoring and Analysis, Springer, 123–147, 2013.

[2] A. Dainotti, A. Pescapé, K. C. Claffy, Issues and future directions in traffic classification, IEEE Network 26 (1) (2012) 35–40.

[3] A. Callado, C. Kamienski, G. Szabó, B. P. Gero, J. Kelner, S. Fernandes, D. Sadok, A survey on internet traffic identification, IEEE Communications Surveys & Tutorials 11 (3) (2009) 37–52.

[4] T. T. T. Nguyen, G. Armitage, A survey of techniques for internet traffic classification using machine learning, IEEE Communications Surveys & Tutorials 10 (4) (2008) 56–76.

[5] G. Aceto, A. Dainotti, W. De Donato, A. Pescapé, PortLoad: taking the best of two worlds in traffic classification, in: IEEE Conference on Computer Communications (INFOCOM) Workshops, 2010, 1–5, 2010.

[6] R. Antonello, S. Fernandes, C. Kamienski, D. Sadok, J. Kelner, I. GóDor, G. Szabó, T. Westholm, Deep packet inspection tools and techniques in commodity platforms: Challenges and trends, Journal of Network and Computer Applications 35 (6) (2012) 1863–1878.

[7] D. Herrmann, R. Wendolsky, H. Federrath, Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial Naïve-Bayes classifier, in: Proceedings of the ACM workshop on Cloud computing security (CCSW), 31–42, 2009.

[8] M. Liberatore, B. N. Levine, Inferring the source of encrypted HTTP connections, in: Proceedings of the 13th ACM conference on Computer and communications security (CCS), 255–263, 2006.

[9] B. Saltaformaggio, H. Choi, K. Johnson, Y. Kwon, Q. Zhang, X. Zhang, D. Xu, J. Qian, Eavesdropping on fine-grained user activities within smartphone apps over encrypted network traffic, in: Proc. USENIX Workshop on Offensive Technologies (WOOT'16, in conjunction with Security'16), 2016.

[10] V. F. Taylor, R. Spolaor, M. Conti, I. Martinovic, Appscanner: Automatic fingerprinting of smartphone apps from encrypted network traffic, in: IEEE European Symposium on Security and Privacy (EuroS&P), 439–454, 2016.

[11] A. Hajjar, J. Khalife, J. Díaz-Verdejo, Network traffic application identification based on message size analysis, Journal of Network and Computer Applications 58 (2015) 130–143.

[12] C.-N. Lu, C.-Y. Huang, Y.-D. Lin, Y.-C. Lai, High performance traffic classification based on message size sequence and distribution, Journal of Network and Computer Applications 76 (2016) 60–74.

[13] Y.-D. Lin, C.-N. Lu, Y.-C. Lai, W.-H. Peng, P.-C. Lin, Application classification using packet size distribution and port association, Journal of Network and Computer Applications 32 (5) (2009) 1023–1030.

[14] G. Aceto, D. Ciuonzo, A. Montieri, A. Pescapé, Traffic Classification of Mobile Apps through Multi-classification, in: accepted at IEEE Global Communications Conference (GLOBECOM), 2017.

[15] L. Bernaille, R. Teixeira, I. Akodkenou, A. Soule, K. Salamatian, Traffic classification on the fly, ACM SIGCOMM Computer Communication Review 36 (2) (2006) 23–26.

[16] L. Bernaille, R. Teixeira, K. Salamatian, Early application identification, in: ACM CoNEXT conference, 6, 2006.

[17] A. Este, F. Gringoli, L. Salgarelli, On the stability of the information carried by traffic flow features at the packet level, ACM SIGCOMM Computer Communication Review 39 (3) (2009) 13–18.

[18] L. Peng, B. Yang, Y. Chen, Z. Chen, Effectiveness of statistical features for early stage internet traffic identification, International Journal of Parallel Programming 44 (1) (2016) 181–197.

[19] A. Dainotti, A. Pescapé, C. Sansone, Early classification of network traffic through multi-classification, in: International Workshop on Traffic Monitoring and Analysis (TMA), Springer, 122–135, 2011.

[20] W. De Donato, A. Pescapé, A. Dainotti, Traffic identification engine: an open platform for traffic classification, IEEE Network 28 (2) (2014) 56–64.

[21] H. He, C. Che, F. Ma, J. Zhang, X. Luo, Traffic classification using ensemble learning and co-training, in: WSEAS Proceedings of the 8th conference on Applied informatics and communications (AIC), 2008.

[22] G. Szabo, I. Szabo, D. Orincsay, Accurate traffic classification, in: IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM), 1–8, 2007.

[23] L. I. Kuncheva, Combining pattern classifiers: methods and algorithms, John Wiley & Sons, 2004.

[24] L. I. Kuncheva, J. J. Rodríguez, A weighted voting framework for classifiers ensembles, Knowledge and Information Systems 38 (2).

[25] S. Dai, A. Tongaonkar, X. Wang, A. Nucci, D. Song, Networkprofiler: Towards automatic fingerprinting of Android apps, in: Proceedings of IEEE INFOCOM, 809–817, 2013.

[26] A. Tongaonkar, A Look at the Mobile App Identification Landscape, IEEE Internet Computing 20 (4) (2016) 9–15, ISSN 1089-7801.

[27] T. Stöber, M. Frank, J. Schmitt, I. Martinovic, Who do you sync you are? smartphone fingerprinting via application behaviour, in: Proceedings of the 6th ACM conference on Security and privacy in wireless and mobile networks (WISEC), 7–12, 2013.

[28] Q. Wang, A. Yahyavi, B. Kemme, W. He, I know what you did on your smartphone: Inferring app usage over encrypted data traffic, in: IEEE Conference on Communications and Network Security (CNS), 433–441, 2015.

[29] V. F. Taylor, R. Spolaor, M. Conti, I. Martinovic, Robust smartphone app identification via encrypted network traffic analysis, IEEE Transactions on Information Forensics and Security .

[30] H. F. Alan, J. Kaur, Can Android Applications Be Identified Using Only TCP/IP Headers of Their Launch Time Traffic?, in: Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks, ACM, 61–66, 2016.

[31] W. M. Shbair, T. Cholez, J. Francois, I. Chrisment, A multi-level framework to identify HTTPS services, in: IEEE/IFIP Network Operations and Management Symposium (NOMS), 240–248, 2016.

[32] T. Bakhshi, B. Ghita, On Internet Traffic Classification: A Two-Phased Machine Learning Approach, Journal of Computer Networks and Communications .

[33] M. Conti, L. V. Mancini, R. Spolaor, N. V. Verde, Analyzing android encrypted network traffic to identify user actions, IEEE Transactions on Information Forensics and Security 11 (1) (2016) 114–125.

[34] M. Shen, M. Wei, L. Zhu, M. Wang, Classification of Encrypted Traffic with Second-Order Markov Chains and Application Attribute Bigrams, IEEE Transactions on Information Forensics and Security .

[35] H. Falaki, D. Lymberopoulos, R. Mahajan, S. Kandula, D. Estrin, A first look at traffic on smartphones, in: Proceedings of the 10th ACM SIGCOMM conference on Internet measurement, ACM, 281–287, 2010.

[36] L. Breiman, Random forests, Machine learning 45 (1) (2001) 5–32.

[37] R. Alshammari, A. N. Zincir-Heywood, Can encrypted traffic be identified without port numbers, IP addresses and payload inspection?, Computer networks 55 (6) (2011) 1326–1350.

[38] Y. S. Huang, C. Y. Suen, A method of combining multiple experts for the recognition of unconstrained handwritten numerals, IEEE Transactions on Pattern Analysis and Machine Intelligence 17 (1) (1995) 90–94.

[39] D. M. J. Tax, R. P. W. Duin, M. V. Breukelen, Comparison between product and mean classifier combination rules, in: Proc. Workshop on Statistical Pattern Recognition, Prague, Czech, 1997.

[40] G. Fumera, F. Roli, Performance analysis and comparison of linear combiners for classifier fusion, Structural, Syntactic, and Statistical Pattern Recognition (2002) 47–64.

[41] J. A. Benediktsson, J. R. Sveinsson, O. K. Ersoy, P. H. Swain, Parallel consensual neural networks, IEEE Transactions on Neural Networks 8 (1) (1997) 54–64.

[42] Q. Liu, Z. Liu, A comparison of improving multi-class imbalance for internet traffic classification, Information Systems Frontiers 16 (3) (2014) 509–521.

[43] L. Peng, H. Zhang, Y. Chen, B. Yang, Imbalanced traffic identification using an imbalanced data gravitation-based classification model, Computer Communications 102 (2017) 177–189.

[44] N. V. Chawla, K. W. Bowyer, L. O. Hall, W. P. Kegelmeyer, Synthetic Minority Over-sampling Technique, Journal of Artificial Intelligence Research 16 (2002) 321–357.

[45] I. H. Witten, E. Frank, Data Mining: Practical Machine Learning Tools and Techniques, 2nd ed., Morgan Kaufmann, 2005.

[46] A. Panchenko, L. Niessen, A. Zinnen, T. Engel, Website fingerprinting in onion routing based anonymization networks, in: Proceedings of the 10th annual ACM workshop on Privacy in the electronic society, ACM, 103–114, 2011.

[47] D. Tammaro, S. Valenti, D. Rossi, A. Pescapé, Exploiting packet-sampling measurements for traffic characterization and classification, International Journal of Network Management 22 (6) (2012) 451–476.

Table 6: F-measure (%) of hard combiners as function of the pool of selected classifiers considering Android (iOS) traffic. Highlighted values: **maximum per pool**, *maximum per combiner*, overall maximum.

Pool of classifiers							Combiners					
Her_Pure	Her_TF	Her_Cos	Lib_NB	Tay_RF	Tay_SVC	CART	MV	WMV	REC	NB	BKS	WER
✓	✓	✓	✓	✓	✓	✓	76.1 (73.8)	**76.7 (73.9)**	76.1 (73.6)	74.7 (72.3)	*75.7 (72.7)*	73.4 (70.2)
✓	✓	✓	✓	✓		✓	75.7 (72.8)	**76.4 (73.3)**	75.5 (72.7)	74.7 (71.8)	74.8 (70.4)	73.3 (69.8)
✓	✓	✓	✓	✓			73.1 (69.7)	**73.8** (70.1)	73.0 (69.9)	72.7 (**70.9**)	71.4 (68.3)	***73.8*** (70.6)
	✓	✓	✓	✓			76.3 (73.2)	**76.7** (73.8)	76.3 (**73.9**)	74.8 (72.7)	73.0 (69.9)	73.6 (70.4)
		✓	✓	✓			*77.5* ***(77.6)***	*77.2 (76.0)*	*77.7 (77.0)*	*75.7 (75.7)*	70.0 (67.8)	72.9 (69.7)
		✓		✓			75.1 (73.2)	75.1 (73.2)	**75.4 (73.6)**	74.4 (74.0)	71.5 (69.0)	73.1 *(70.9)*

Table 7: F-measure (%) of soft combiners as function of the pool of selected classifiers considering Android (iOS) traffic. Highlighted values: **maximum per pool**, *maximum per combiner*, overall maximum.

(a) Class-conscious (CC) non-trainable combiners (1).

Pool of classifiers							Combiners				
Her_Pure	Her_TF	Her_Cos	Lib_NB	Tay_RF	Tay_SVC	CART	Mean	Maximum	Minimum	Median	PP
✓	✓	✓	✓	✓	✓	✓	73.8 (70.9)	55.0 (50.3)	39.3 (36.1)	**76.9 (74.3)**	39.5 (36.5)
✓	✓	✓	✓	✓		✓	73.2 (70.4)	55.2 (50.7)	39.6 (36.1)	74.6 (72.2)	39.6 (36.3)
✓	✓	✓	✓	✓			70.4 (67.7)	62.3 (60.9)	52.0 (51.3)	71.5 (**69.1**)	55.6 (55.1)
	✓	✓	✓	✓			75.7 (73.1)	71.6 (70.0)	64.0 (61.6)	76.3 (73.7)	67.5 (66.3)
		✓	✓	✓			74.8 (73.9)	71.1 (70.2)	64.7 (63.0)	76.6 (**75.3**)	67.9 (66.3)
		✓		✓			*77.5 (74.1)*	*76.6 (72.8)*	*79.0 (75.2)*	*77.5 (74.1)*	*75.8 (73.1)*

(b) Class-conscious (CC) non-trainable combiners (2).

Pool of classifiers							Combiners			
Her_Pure	Her_TF	Her_Cos	Lib_NB	Tay_RF	Tay_SVC	CART	Trim Mean	Harm Mean	Geom Mean	Gen Mean
✓	✓	✓	✓	✓	✓	✓	76.2 (73.5)	39.3 (35.9)	39.6 (36.5)	75.7 (73.2)
✓	✓	✓	✓	✓		✓	**75.5 (72.7)**	39.5 (36.2)	39.8 (36.1)	74.6 (72.2)
✓	✓	✓	✓	✓			**71.9** (69.0)	52.3 (51.8)	58.7 (56.5)	71.3 (**69.1**)
	✓	✓	✓	✓			76.3 (73.7)	65.2 (62.1)	72.1 (68.2)	**76.6 (74.4)**
		✓	✓	✓			**76.7** ***(75.3)***	65.1 (63.5)	71.5 (67.8)	76.3 *(75.7)*
		✓		✓			*77.5* (74.1)	*79.2 (75.7)*	***79.6 (76.8)***	*78.8* (75.3)

(c) Class-conscious (CC) trainable combiners.

Pool of classifiers							Combiners		
Her_Pure	Her_TF	Her_Cos	Lib_NB	Tay_RF	Tay_SVC	CART	Fuzzy Integral	K weights	KL weights
✓	✓	✓	✓	✓	✓	✓	*75.6 (72.2)*	75.2 (71.7)	**79.1** ***(76.8)***
✓	✓	✓	✓	✓		✓	70.5 (70.4)	72.1 (69.2)	***79.4 (76.8)***
✓	✓	✓	✓	✓			72.0 (69.4)	73.5 (69.8)	**79.3 (76.4)**
	✓	✓	✓	✓			72.3 (70.7)	74.9 (72.9)	***79.4*** **(76.5)**
		✓	✓	✓			72.9 (71.0)	76.3 *(73.6)*	**78.8 (76.2)**
		✓		✓			71.6 (69.3)	*76.7* (73.1)	**78.2 (75.1)**

(d) Class-Indifferent (CI): Decision Templates (DT) and Dempster-Shafer (DS) approaches.

Pool of classifiers							Combiners			
Her_Pure	Her_TF	Her_Cos	Lib_NB	Tay_RF	Tay_SVC	CART	DT-SE	DT-L1	DT-FSD	DS-L2
✓	✓	✓	✓	✓	✓	✓	73.1 (69.1)	72.8 (**70.7**)	72.5 (69.3)	**73.2** (69.7)
✓	✓	✓	✓	✓		✓	73.1 (69.1)	72.8 (**70.8**)	72.4 (69.3)	**73.2** (69.5)
✓	✓	✓	✓	✓			**70.5** (67.7)	68.8 (66.9)	68.1 (65.5)	70.2 (**67.8**)
	✓	✓	✓	✓			**73.1 (70.6)**	72.5 (69.3)	72.2 (70.1)	72.8 (70.2)
		✓	✓	✓			*73.7 (72.0)*	***74.1*** *(71.5)*	73.3 ***(72.2)***	*73.9 (71.7)*
		✓		✓			73.3 (**70.3**)	**73.9** (70.2)	*73.7* (69.5)	73.2 (70.1)

Table 8: *Maximum Improvement Over Best Classifier* (MIOBC) of the F-measure (%), as function of the pool of selected classifiers, for each class of hard and soft combiners, considering Android (iOS) traffic.

Pool of classifiers							Combiners			
Her_Pure	Her_TF	Her_Cos	Lib_NB	Tay_RF	Tay_SVC	CART	Hard	CC non-trainable	CC trainable	CI
✓	✓	✓	✓	✓	✓	✓	+4.4 (+4.5)	+4.6 (+4.9)	+6.8 (+7.4)	+0.9 (+1.3)
✓	✓	✓	✓	✓		✓	+4.1 (+3.9)	+3.2 (+3.3)	+7.1 (+7.4)	+0.9 (+1.4)
✓	✓	✓	✓	✓			+1.5 (+1.5)	*-0.4 (-0.3)*	+7.0 (+7.0)	*-1.8 (-1.6)*
	✓	✓	✓	✓			+4.4 (+4.5)	+4.3 (+5.0)	+7.1 (+7.1)	+0.8 (+1.2)
		✓	✓	✓			+5.4 (+8.2)	+4.4 (+6.0)	+6.5 (+6.8)	+1.8 (+2.8)
		✓		✓			+3.1 (+4.2)	+7.3 (+7.4)	+5.9 (+5.7)	+1.6 (+0.9)