I Know What You Did On Your Smartphone: Inferring App Usage Over Encrypted Data Traffic

Qinglong Wang
McGill University
qinglong.wang@cs.mcgill.ca

Amir Yahyavi
Carnegie Mellon University
ayahyavi@andrew.cmu.edu

Bettina Kemme
McGill University
kemme@cs.mcgill.ca

Wenbo He
McGill University
wenbohe@cs.mcgill.ca

***Abstract*—Smartphones and tablets are now ubiquitous in many people's lives and are used throughout the day in many public places. They are often connected to a wireless local area network (IEEE 802.11 WLANs) and rely on encryption protocols to maintain their security and privacy. In this paper, we show that even in presence of encryption, an attacker without access to encryption keys is able to determine the users' behavior, in particular, their app usage. We perform this attack using packet-level traffic analysis in which we use side-channel information leaks to identify specific patterns in packets regardless of whether they are encrypted or not. We show that just by collecting and analyzing small amounts of wireless traffic, one can determine what apps each individual smartphone user in the vicinity is using. Furthermore, and more worrying, we show that by using these apps the privacy of the user is more at risk compared to using online services through browsers on mobile devices. This is due to the fact that apps generate more identifiable traffic patterns. Using random forests to classify the apps we show that we are able to identify individual apps, even in presence of noise, with great accuracy. Given that most online services now provide native apps that may be identified by this method, these attacks represent a serious threat to users' privacy.**

Keywords*—*Wireless, side-channel attack, privacy, smartphone applications, machine learning, random forest.

I. Introduction

Currently 167.9 million people in the U.S. own smartphones (69.6% of mobile subscribers) [3] and 42% own tablets [17] and the numbers are increasing fast. In US, mobile devices have overtaken the PCs' internet usage and now they make up 55% of the internet usage [3]. Most of this traffic is generated by apps as they have become more and more popular on smarthphones: by 2013, more than 60 billion iOS apps have been downloaded and both Apple app store and Google play store have over one million apps available [17]. Popular apps like Facebook are used by a large number of users (74.1%) [3]. These applications provide native support for a wide range of services like browsing, chatting, social networks, streaming audio and video, etc., and offer a faster and simpler interface than their web-based counterparts.

Given that mobile devices rely on wireless connectivity, they can be highly susceptible to security and privacy risks as malicious attacks are more severe in wireless, e.g., wireless local area networks (WLANs) than wired environments.These networks typically employ encryption technologies such as WEP, WPA, and WPA2 to prevent unauthorized access to the network traffic. However, these techniques cannot protect the traffic from all types of attacks. In [25], for example, the authors, through analytical techniques, infer the general type of online activity performed, such as browsing and social networks. Traffic analysis can achieve this by employing packet-level statistical information (MAC layer) such as average packet sizes, packet size distributions, and average inter-arrival times to find identifiable patterns of traffic.

While knowing what kind of activity a user might perform is already useful for all kinds of malicious behavior, we argue that knowing the specific apps used is an even more serious privacy & security concern. For privacy, in its simplest form, an attacker simply aims to monitor a victim's activities. Some apps such as health monitoring apps and dating apps can be particularly privacy sensitive. Detailed information about app usage can also be used to design more targeted attacks. For example, after discovering that a victim uses a particular job hunting or health app, the attacker may issue phishing attacks by emailing the victim and mentioning fake job opportunities or a potential treatment to a certain disease. For security, although wireless communication reliability can be effectively improved [18], according to [10], Android Virus Detectors (AVDs) are nullified by the android system during engine update, leaving the device exposing to severe threats. A malicious attacker can take advantages of this vulnerability more efficiently if this attacher is aware of the types of AVDs utilized by the user. Moreover, home security systems controlled by smartphones are in threat of leaking information about the presence of the owner to attackers, and may even fail in alarming the owner when attacked by a malwares [26]. Finally, many companies are already tracking customer's movements and behavior based on the MAC address of their smartphones and their credit card usage. A more severe case might be an insurance company collecting information about users using a specific health monitoring app which in turn can affect the premium they pay.

In this paper, we show that it is possible to determine the particular apps a victim is using even if encryption is in place and the attacker has no access to the encryption keys. We perform this attack using packet-level traffic analysis in which we use side-channel information leaks to identify specific patterns in packets regardless of whether encryption is used or not. We argue that using apps on mobile devices generates distinct and identifiable traffic patterns resulting in serious privacy concerns. We also compare app usage with using the same services through a web browser. While traffic generated by an individual is quite unique and thus, traffic from different apps have a good potential to be distinguishable from each other, we show that using a service through a browser can be more blurry and does not generate the same traffic patterns as the corresponding apps. However, by being smart of how we train web-browser activity, we are able to also detect the corresponding services used. Another contributing factor, that makes detection of user activity on smartphones and tablets easier, is that unlike desktop computers, most smartphone and tablet users work with one app at a time and multi-tasking is fairly limited. Furthermore, to infer usage patterns, we show that only a short-term analysis of the traffic is sufficient.

In particular, this paper uses machine learning techniques based on random forest (RF) to determine app usage of mobile users. We propose a mechanism to build the forest based on training data that contains packet-level statistical information of a predefined set of applications. By using this technique we are able to classify known apps with considerable accuracy. We also have high true positives for each individual app. For the purpose of this paper we use a wide range of applications, covering various categories including browsing, gaming apps, social networking, multimedia, dating, financial and medical.

In summary, the contributions of this paper are as follows:

- We show that inferring behavioral patterns on s-martphones and tablets based on packet-level traffic analysis of encrypted messages is possible and only requires logging the traffic for a short period of time. No vulnerability in the encryption or the OS is needed to derive the information.
- We infer the exact apps a user is utilizing (vs. simply determining the general category of use like browsing) which leads to significant security concerns, as such detailed information can be used for more severe attacks.
- Our approach is based on random forest and it is able to detect a wide range of apps with good accuracy.

The rest of this paper is organized as follows: Section II covers the related work. In Section III we introduce the traffic features used for our meachine learning approach as well as describe the apps we consider. Section IV presents an evaluation of our solution, including an analysis of the accuracy and true positive rates in our system as well as the various features of the different apps considered. We also consider the case of accessing content through browsing and the existence of noise. We further provide potential countermeasure for attacks taking advantages of side-channel information leaks, conclude the paper in Section VI.

II. Related Work

Most of today's cellular and wireless networks use encryption to protect users' privacy, making it substantially more difficult for attackers to violate users' privacy via decrypting their messages. However, by using patterns in the encrypted traffic, one can infer private user activities such as web browsing [12, 20], spoken phrases [22], motion and location [11, 14], identification [19], and behaviors [4, 25]. The features of encrypted traffic used for privacy attacks are often referred to as *side-channel information leaks* [2].

Study of side-channel attacks specific to smartphones and tablets has been fairly limited in research on side channel information leakage. Built-in sensors in smartphones have been utilized to extract usage patterns of applications [7]. Mobile devices have to rely on wireless communications making them particularly vulnerable to wireless traffic analysis attacks. 3G side-channel information has been used to reliably identify smartphone usage [19]. In addition, data caps, speed limits, and coverage problems often make most mobile users to prefer WLANs, when they are available. WLANs are similarly vulnerable to traffic analysis attacks [25].

Inferring behavioral patterns of users in different contexts depends on a range of techniques to reveal distinctive user patterns according to the gender, economical, and personal habits [4, 19, 25]. These techniques generally rely on using the statistical features contained in the encrypted traffic, to identify users' identity or their behaviors. However, there exist immense diversity in users' smartphone usage patterns [9] making the identification complicated. More concretely, people use smartphones at different times during the day, with different frequency and duration, and conduct different interactions with different web services. Therefore, designing a scheme suitable for analyzing all users' smartphone usage behavior will be impractical. However, when viewing from the app side, the diversity doesn't hold since for each single action when using a particular app, the statistical patterns do not show great diversity. Research on side-channel information analysis shows promise for handling this case.

[4] utilizes side-channel information leaks to infer users' actions when using three popular smartphone apps. Since their work is targeted on encrypted TCP/IP traffic, the authors can identify the traffics generated for each specific apps based on the source and destination IP addresses, especially by reverse resolution to identify the web servers. The analysis is carried on TCP sessions, as a result, the statistical features of the transmitted packets can be well discovered given that there is little noise within each session. However, the authors only tackle a few different actions corresponding to a limited number of apps. Considering the countless number of actions in apps in today's online stores and their diversity, this work cannot be easily scaled to a much larger number of app usage analysis. Research work in [19] uses a similar approach to generate device fingerprints for smartphone users by collecting and analyzing the traffics generated by several popular apps. The authors use a burst of traffic as the fundamental entity, which can be extracted and divided into samples using a predefined time length threshold. Although this separation can handle the diversity in app usage, this approach cannot infer the correlation of successive packets, given that most users use smartphone apps continuously. In [25], a wireless MAC-based approach has been developed for inferring laptop users' online activities regardless of the encryption techniques used in the WiFi network. Machine learning methods are used to classify eight common online activities.

Defending against these attacks can be hard and expensive. For example, in order to thwart these attacks [24] proposes employing virtualized MAC addresses to confuse the adversary's analysis. However, the solution is more suitable for using a WiFi connection on a PC (further discussed in Section V).

It is a common practice to pre-process the collected traffic from smartphone apps. The pre-processing procedures such as dynamic time wrapping (DTW) method [4, 14], sliding window [25] measure the similarity, mutual information, or correlations between traffic samples in order to extract more hidden features or eliminate irrelevant features. When it comes to selecting classification methods, classical machine learning methods are more commonly employed (e.g., neural networks, support vector machines, Bayesian techniques, and Hidden Markov Models [5, 15, 25]). Since traffics collected are encrypted, they cannot be easily separated according to their sources or destinations IP address. As a result, more noise can be expected to be mixed in the collected traffics, in term of traffic belonging to concurrent and irrelevant apps (e.g. apps running at the background that generate syncing message or continuously stream data). Hence, the classification methods selected should also be capable of handling rather noisy traffic logs. Random forest (RF) [4, 21] has been recently utilized for this purpose and similar to [4, 21], in our work, we also employ random forest for classification. Additionally, we exploit RF for further evaluating the importance of extracted features, which can shed a light on potential countermeasures for traffic analysis attacks based on side-channel information leaks.

III. Design of Architecture

A. Adversary Model

We consider a common scenario where a malicious attacker aims to infer personal online activities of users. The attacker sniffs the encrypted traffic on the same WLAN channel as the access point (AP) to collect the traffic necessary to perform further pattern recognition analysis (Fig. 1). This type of attack can be applied to most encrypted wireless environments, e.g., a neighbor's WLAN, coffee-shops' networks, *etc*. In order to perform the attack against a known user, we assume that we should have the user device's MAC address. However, if not, the MAC addresses from collected traffic may be looked up in organizationally unique identifier (OUI) lists, which would help identify the user's device and the user himself. Furthermore, wireless localization techniques can also determine location of the user and identify him [16].

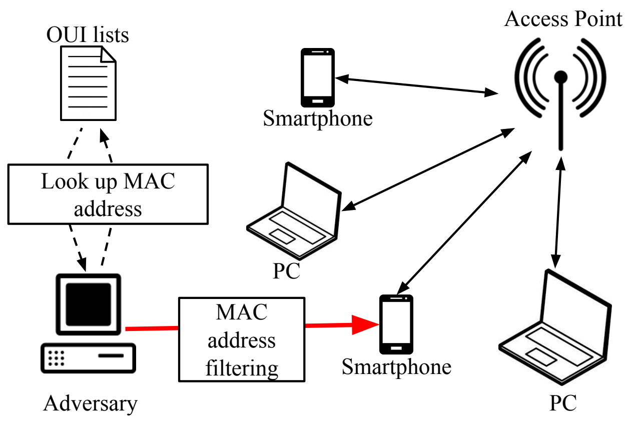

Fig. 1. Common Adversary Model of Side-channel Information Leaks. Assume that the adversary locate within the same area as the target does, where is covered by the same WLAN network. Hence, the adversary can utilize the OUI list to match the collected MAC address and the target's device, then filter out irrelevant MAC addresses.

B. Wireless Traffic Collection

An attacker can gain substantial information if he has access to the same WLAN network as the target does, therefore, we assume that attacks based on side-channel information leaks are only used when the adversary does not have access to the encrypted WLAN. In this paper, we collect data from different WPA2-encrypted environments like at home or campus. WPA and WEP offer weaker forms of encryptions and would be similarly susceptible to these attacks. While here we focus on WLANs, we believe similar attacks should be possible in cellular networks with relative ease [19] as well. The wireless card is put in monitor mode to collect all the wireless traffic between hosts and the access point and a popular sniffing tool like *Wireshark* or *aircrack-ng* is used to collect the traffic and perform network protocol analysis. In order to perform the classification analysis the attacker filters the traffic collected according to the target's MAC address.

1) App Selection: We consider apps from iOS (with preliminary results from Android). Since both ecosystems heavily rely on apps to provide many of the main functionalities of the smartphone. In an attempt to ensure that we cover a wide range of application types, we decides to select some of the most popular apps covering a wide range of different app categories including browsing, games, multimedia, online chat, and social networks (see Table I). Thus, we show that our attack can effectively detect a large part of current application activity. However, we believe that the approach can be used with any set of particular apps that would be of interest. In fact, we believe that an attacker is most likely interested in a few privacy sensitive apps and does not require a comprehensive tool to analyze all the apps that exist in app stores. We further discuss the impact of noise of apps unknown to the attacker in Section IV-D.

Note that in this paper we focus on detecting individual app use and not whether the user uses any kind of app within a certain category. We believe that while determining a category might be beneficial by itself, determining individual apps provides more power to the attacker. Furthermore, it might actually not be that easy to just determine the category. Different apps in the same category can have very different traffic patterns. For example, Twitter and Facebook both belong to the social network category, however, given the different content they receive from web servers, one being mostly small text messages and the other containing many pictures, the resulting traffic pattern is very different. Therefore, an analysis on an apps-basis is more appropriate. Detecting new apps from the same category as a known app is not necessarily possible and each app has to be individually analyzed. For any other apps out of the scope of our consideration, the system should ideally indicate that it is an unknown app.

2) Data Set and Feature Extraction: We run each app of interest for sufficient monitoring time (300 seconds, in Section IV-B) and collect the traffic information that is then fed into our machine learning approach. This analysis learns each app's traffic pattern, *signature*, in order to maximize the accuracy of the detection.

When using an app, each action, e.g. publishing a post on Facebook, loading a video on Youtube, or messaging via Facebook Messager will initiate a sequence of traffic flows

TABLE I. ACTIVITY CATEGORIES, CORRESPONDING APPS AND ACTIONS

Categories	Applications	Actions
Browsing	Chrome(CH)	Loading text, pictures and streaming
Gaming	Boom Beach(BB)	Gaming action
Multimedia	YouTube (UTB), Songza(SON)	Streaming
Online Chatting	Facebook Messenger (FBM), Tecent QQ (QQ), Snapchat (SN)	Sending and receiving text, pictures
Social Network	Facebook (FB), Twitter (TW)	Posting, messaging, adding contact, loading text, pictures
Dating	Tinder (TD)	Loading pictures
Financial	Mint (MT)	Configuring account, loading text, pictures
Medical	CDC News (CDC), Medscape (MED)	Loading text, pictures

between the source (e.g. a smartphone) and destination devices (e.g. a switch). This sequence of traffic flows can be represented by a burst, which consists of many frames aggregated densely within a certain observation period, as shown and discussed in Fig. 2. These bursts are often generated by the app that is currently being used rather than background processes. Therefore, in our approach, we only use messages during bursts when we prepare our initial training set. Similarly, when we run the analysis on the test datasets, we first clean that dataset to only contain messages within bursts. We believe that with this we can effectively reduce the noise generated by apps that run in the background.

Each burst collected is comprised of WPA2-encrypted frames. In order to extract more samples for future analysis, as well as better exploration of the statistical features hidden within the frames, we employ a sliding window scheme that narrows down the observation period to a much smaller scale. Assume that a burst contains N frames, and we set the window size to be W. Then we can extract $N - W + 1$ samples: $\{1, \ldots, W\}, \{2, \ldots, W + 1\}, \ldots \{N - W + 1, \ldots, N\}$. The traffic frames contained in each window are referred to as '*samples*', to differentiate them from the original burst.

Using this larger number of extracted traffic samples, we further pre-process the samples to obtain additional statistical features hidden in the original logs: frame arrival time, frame size, and direction (receiving or transmitting). We calculate different distributions of frame sizes from both sending and receiving transmissions. For each distribution of frame sizes, we further obtain the average frame sizes, and the overall deviation of frame sizes of this sample. We also calculate the inter-arrival time between sequential frames for both sending and receiving traffic. Similar to frame sizes, we compute the average inter-arrival time for different distributions, and the average and deviation of overall inter-arrival time of this sample. Hence, in total, we have twenty features, ten of which are for transmitting(Tx) traffic and the other ten are for receiving(Rx) traffic. Table II lists all the extracted features.

TABLE II. EXTRACTED FEATURES FROM COLLECTED FRAMES

Number(Tx)	Features (Tx)	Number(Rx)	Features (Rx)
①	Ave size	⑪	Ave size
②	Ave size (low 20%)	⑫	Ave size (low 20%)
③	Ave size (mid 60%)	⑬	Ave size (mid 60%)
④	Ave size (high 20%)	⑭	Ave size (high 20%)
⑤	STD size	⑮	STD size
⑥	Ave time	⑯	Ave time
⑦	Ave time (low 20%)	⑰	Ave time (low 20%)
⑧	Ave time (mid 60%)	⑱	Ave time (mid 60%)
⑨	Ave time (high 20%)	⑲	Ave time (high 20%)
⑩	STD time	⑳	STD time

TABLE III. EXTRACTED FEATURES FROM COLLECTED FRAMES

MAC address	Company	OS
24:E3:14	Apple	iOS
24:F5:AA	Samsung	Android
00:1A:11	Google Nexus	Android
00:09:2D	HTC	Android

C. Classification

An adversary needs information about the operating system the victim is using before conducting effective traffic analysis. As the adversary can identify the target via MAC address filtering, he can further use the OUI list to guess the type of device and operating system prior to using the classifier. Then the attacker can train different classifiers for different operating systems and use them according to the victims' OS. Examples of MAC address prefixes for several popular devices, and their OSes are shown in Table III.

We employ random forests (RF) as the classification method [1] using the open-source library scikit-learn. The idea is to build a forest of uncorrelated decision trees for classification (each class being an app). Given as the traffic of an application to classify, the forest outputs the class that is the mode of the individual trees, that is, the class that is returned the most often over all decision trees in the forest. The set of decision trees are built by selecting, at each candidate split, a random subset of the features. It also considers different sub-samples of the training data when generating the trees. RF have a set of properties that make them particularly useful for our classification task: (1) RF is an ensemble learning method which can handle situations where substantial noise exists in the dataset [6]. (2) Both variance and bias are somewhat mitigated by using RF, given that RF averages the results of all aggregated decision trees, which is known to have a low bias. (3) RF is capable of providing evaluation of feature importance, therefore, it not only eliminates the necessity of feature selection, but also helps better understand the significance of various statistical features. (4) RF provides an internal unbiased out-of-bag (OOB) estimate [8] of the classification performance on the fly. Thus the dataset can be better utilized as a whole without splitting it as commonly required by cross validation. (5) RF is faster than both bagging and boosting, especially when using parallelization. In our experiments, generating a forest with 30 trees took less than a second.

IV. EXPERIMENTS

A. Experiment Setup

In this section, we present an extensive performance evaluation, looking at many aspects of classification performance. First, we analyze the influence of the many parameters in our configuration. We look how feature selection and importance as well as the number of estimators (i.e., decision trees) can influence performance. We also look at the impact of data set pre-processing parameters such as the monitoring time T and the window size W. Next, we discuss how services can be

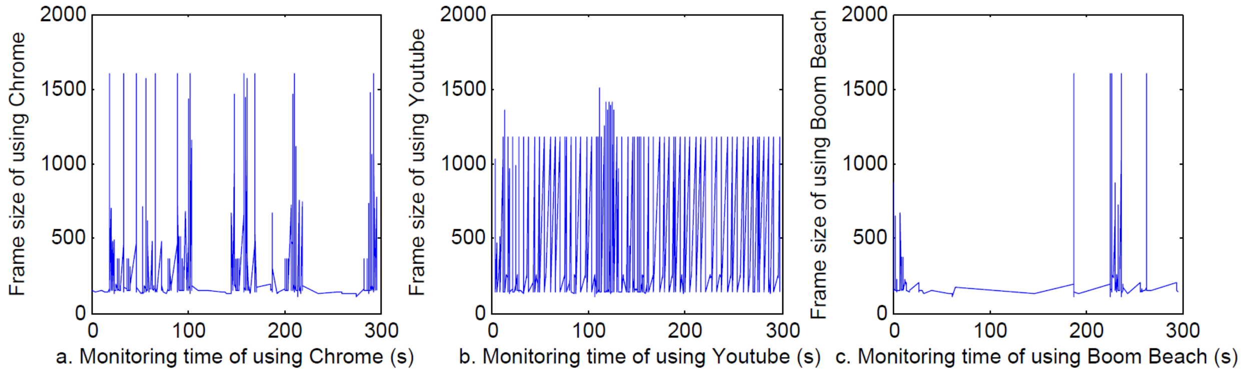

Fig. 2. Burst examples from using Chrome, Youtube and Boom Beach. The burst density for using Chrome is medium, which is closely related to the frequency of conducting an action that will trigger a sequence of traffic flows. While burst density for using Youtube is considerably high, this can be attributed to its streaming features when the loading of an online video is initiated. The burst density for Boom Beach can be regarded as sparse. This gaming app appears to initiate traffic exchange rather infrequently. Note that the largest frame sizes are at 1500 Byte. In Chrome and Boom Beach, various frame sizes are used, while in Youtube, a large percentage of messages has identical frame size. Therefore, an analysis that takes advantage of statistical features including frame size, frequency of burst generation as well as the duration of intense and sparse bursts can be beneficial for effective traffic inference.

correctly determined whether they are accessed through an app or through a web-browser. Finally we analyze the influence of noise, that is what happens when multiple apps are running simultaneously or there exist unknown apps.

The network environment in which we conducted the experiments: (1) supports 802.11a/b/g modes, (2) is encrypted by WPA2, and (3) maintains a stable rate varying from 48Mbps to 54Mbps. In this section we focus on our results with iOS, motivated by that fact that according to [13], iOS users perform considerably more often online activities. In this section, we describe the experiment results obtained on iOS systems to show the effectiveness of our approach to infer app usage.

As classification performance metrics, we use the true positive percentage of each class investigated, the overall accuracy and the OOB estimate accuracy. True positive is the percentage of traffic samples of a given class X that is correctly classified to class X. The overall accuracy is the percentage of correctly classified samples among all samples tested. The OOB estimate accuracy is the unbiased estimate of the classification performance obtained via bagging, which is utilized by the RF method.

To collect the dataset, we randomly perform various actions for all 13 apps presented in Table I. This reduces the chances of relying on one user's particular behavior pattern. The data set is then split into a training set and a testing set with a ratio of $6/4$. As RF has a fair amount of randomness in selecting features, the classification results might vary from time to time. Thus, our figures show the average results of 1000 experiments.

B. Classification Performance

We analyze the classification performance under different parameter configurations. The initial setup is as follows. The monitoring time, that is the time we run each application and collect the traffic, is 300 seconds. Window size is $W = 10$. There are 20 features as shown in Table II, and the number of estimators (decision trees) is set to $N_e = 30$. The classification performance of this initial setting is shown in the black histograms in Fig. 3. The following subsections then all look at the performance when one of these parameters is changed.

In Fig. 3 (black columns), the first column shows the OOB estimate accuracy, the second shows the overall accuracy denoted as OA. The remaining columns show the true positives for each class, denoted by the abbreviations used in Table I. The OOB estimate accuracy and the overall accuracy are 92.37% and 93.96%, respectively. This result demonstrates the effectiveness of using RF to classify the traffic generated by smartphone apps. The worst true positive percentage obtained is 87.23% for classifying samples generated by Boom Beach. We assume that because Boom Beach generates traffic frames with relatively low frequency and limited size, it is more difficult to remove noise from the training and testing samples. But overall, the results are very promising.

1) Feature Importance: The 20 features we extract reflect different aspects of the traffic samples (e.g. feature ① - ⑤ and ⑪ - ⑮ are for frame sizing and the rest are for inter-arrival time). Not all these features are equally important regarding their influence on the classification performance. In fact, having too many features might lead to overfitting in the worst case. In this section, we analyze how RF cope with our features.

Fig. 4 shows the feature importance values for the fea-

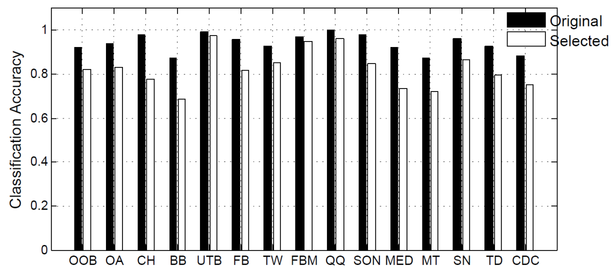

Fig. 3. Classification performance in terms of OOB estimate accuracy, overall accuracy, and true positive for each class, for datasets with complete features (black columns) and selected features (white columns). The lowest true positive values (87.23% with complete features, 68.59% with selected features) are for classifying traffic generated by Boom Beach.

tures we extracted. We demonstrate the averaged (overall) importance value over the data sets of all our apps, as well as three example apps including Boom Beach, Facebook and Youtube. We can see that the different settings differ in their most important features. For example looking at the overall feature importance values, the six most important features are the average size of all frames (feature ①), average size of smallest 20% frames (feature ②), middle 60% frames (feature ③) and largest 20% frames (feature ④) of transmitted data, and the average inter-arrival time of the frames with the lowest 20% intervals in both directions of transmitting (feature ⑦) and receiving (feature ⑰). That is, transmitted frame sizes are more indicative than received frame sizes but for inter-arrival features both directions play a role, but only for the fastest response times. The deviation between significant and insignificant features are relatively large. For insignificant features, 13 (out of 14) features have importance values lower than 0.05. This might lead to the conclusion that eliminating them might not lead to significant performance loss.

To analyze the importance of feature selection, we rerun the classification experiments by only using the overall six most important features. The classification performance is shown in the white histograms in Fig. 3. In fact, for some apps, the performance is nearly the same as when considering the full feature set (e.g, 1.73% difference for Youtube). However, for others, there is a considerable performance loss (up to 21.38% for Boom Beach). The reason is that the average feature importance does not reflect perfectly the importance of certain features for a particular application. In particular, Fig. 4 also shows that that Boom Beach's most important features differ from the average. For instance, overall and for Youtube, frame size is not an important feature of received traffic while it has more importance for Boom Beach.

As a conclusion, we can see that removing features with overall little importance can lead to considerable performance loss. Thus, it is important to keep all features. By using RF and generating many decision trees, we can exploit all the features when needed, and at the same time avoid overfitting for those applications where the features are not relevant.

2) Influence of the Number of Estimators: The number of estimators N_e used when building the RF can influence

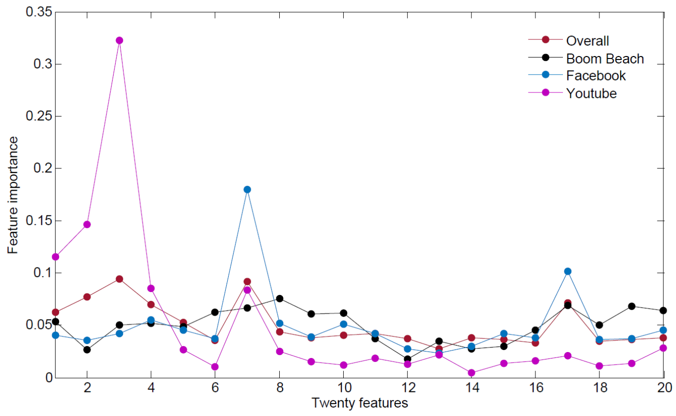

Fig. 4. Feature importance of the overall result, apps including Boom Beach, Facebook and Youtube.

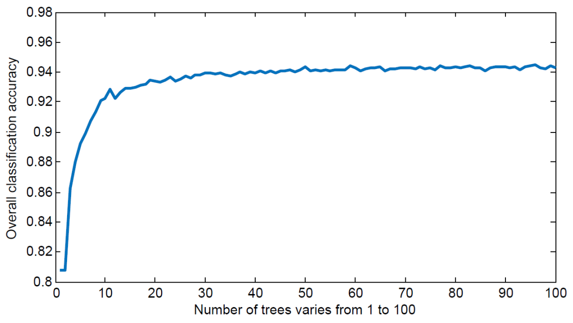

Fig. 5. Influence of number of estimators on overall accuracy. The overall accuracy converges to around 94% when the number of estimators exceeds 20. As a single estimator already provides an accuracy of 80% we can see that decision trees are a good approach for traffic analysis of mobile apps.

the classification performance, in that both the strength of each independent estimator and the correlation between any two estimators are closely related to N_e. The strength and correlation together influence the generalization error, which has been proved to converge to a limit as N_e increase [1].

In this experiment, we vary N_e from 1 to 100, while keeping all other parameters to the base value (20 features, $W = 10$, $T = 300s$). The result is shown in Fig. 5. We can see that the overall accuracy improves rapidly when N_e increases from 1 to 10. When N_e exceeds 20, the overall accuracy starts to approach a stable limit around 94%. This result indicates that a proper selection of N_e can achieve a sufficiently high accuracy, while saving much computational efforts.

3) Influence of Monitoring Time: In this experiment, we explore the influence of monitoring time on the classification performance. We aim at investigating how long we have to observe the app use before we can achieve a high classification accuracy. We set $T = 50s, 100s, 200s, 300s$, and show the results in Fig. 6. Interestingly, while for some apps (e.g. Chrome, Youtube, Facebook), the overall accuracy slightly improves or remains unchanged with increasing T, longer monitoring times have actually a negative impact on other apps (e.g. Mint, Tinder, CDC News). We can explain this by the additional noise introduced by longer monitoring times. This noise appears to be more detrimental for apps that have limited traffic such as Boom Beach, Mint, CDC News, etc.

Thus, we derive from Fig. 6, that $T = 100$ is likely the best monitoring time. Clearly, this is a sensitive parameter and there is no obvious best solution. An attacker might have to adjust the monitoring time depending on the applications he wants to detect. But what becomes clear is that only very little observation time is needed before an adversary is capable of predicting the app traffic with high accuracy. It shows that RF allows for an effective online app usage attack.

4) Influence of Window Size: In the pre-processing step of our approach, we divide the original traffic into more samples using a sliding window of size W. A larger window size means each sample will contain more frames within itself, and therefore has a higher chance of covering frames with various sizes. This might lead to some (burst) features to be less distinctive (e.g. neutralize the sparse bursts of Boom Beach as shown in Fig. 2). Meanwhile, a smaller window narrows the observation to a few frames, possibly leaving the classification more vulnerable to noise.

Thus, in this experiment we vary the window sizes between $W = 5, 10, 20, 30$ to demonstrate its influence on the true positive values of each class. The same W values are used in both training and testing procedures, as W determines the number of frames contained in an aggregated sample. We show the true positive for each class in Fig. 7. We can observe that the true positive values for all classes improve gradually as W increases. At $W = 30$, all apps can be correctly identified with accuracy values of more than 85%.

Therefore, this result implies that when the traffic is aggregated to larger samples, the averaged statistical features are less affected, and as a result the statistical distribution of frames is closer to the real average values. This can be further explained by considering the traffic as time series data. Commonly, a smartphone user operates an app for much more than the window size. Therefore, a larger window size not only covers the statistical features of the current sample, but also captures the correlation between successive samples, therefore the actions related to a single user operation can be more coherently contained in the samples. However, this does not mean that the W can be arbitrarily large. There is negative linear correlation between the number of samples and window size. A larger window size leads to less training samples.

Again, among all thirteen apps, Boom Beach has the lowest true positive percentages. To better interpret the results for Boom Beach, Fig. 8. shows the classification probability distribution for Boom Beach (it shows for each app, the probability that a Boom Beach test data set is classified to be one of these apps). Although a data set is correctly classified as being Boom Beach less than 50% of the time when $W = 5$, the probability for recognizing the traffic as originated from Boom Beach still dominates the distribution. Again, the background noise might overshadow the actual Boom Beach traffic patterns. When W increases from 5 to 30, the true positive percentages reaches 86.98%.

C. *Browser vs App*

Compared to laptop or desktop users, smartphone users tend to use apps instead of accessing the corresponding websites with their browsers. This holds true in particular for social apps such as Twitter and Facebook. In this section, we want to analyze whether the network traffic generated by a service used through a browser is similar to the one generated by using the corresponding app. If yes, our classifier only needs to be trained on the app, and would still correctly classify traffic generated when using the same service through a browser. To do so, we use test traffic generated by accessing Facebook and Twitter through Chrome to test the classifier. The result in term of classification probability distribution for both services is shown in Fig. 9. We can observe that our classifier is able to correctly determine the service to some degree. For Facebook, our classifier correctly assigns the traffic to belong to Facebook with a probability of 30%, and wrongly decides on Songza and Twitter around for 17% of the samples, and less on the other apps. For Twitter the classification is correct up to 40% with the second highest, wrong, classification (YouTube) being around 13%.

We conclude from here, that the traffic, at least for these two services, looks to some degree different depending on whether it is generated by an app or through a web-browser interface. Thus, further measures have to be taken to detect different services used within a web-browser. One option is to handle “using Facebook via the browser” and “using Twitter via a browser” as simply yet further apps whose traffic patterns will be learned via a learning data set and the classes integrated into our RF. Indeed, this approach leads to very good results as shown in Fig. 9. Our test data set on Facebook via the browser is correctly classified with a probably higher than 70%, and and is classified with a probability of more than 10% to be generated by the Facebook app. For Twitter the results are even better.

D. *Influence of Noise*

App traffic can be contaminated by different types of noise. First, a user can use several apps at the same time. For example, a smartphone user can listen to music while using Facebook. Hence, the testing samples will contain messages from different apps, and thus, as a whole, do not follow the traffic pattern of any one of training data sets. Similarly, the traffic might contain flows generated by apps that are not included in the training set when building the classifier.

We don’t think it is possible or desirable to build a classifier that effectively classify all these cases. First of all, there

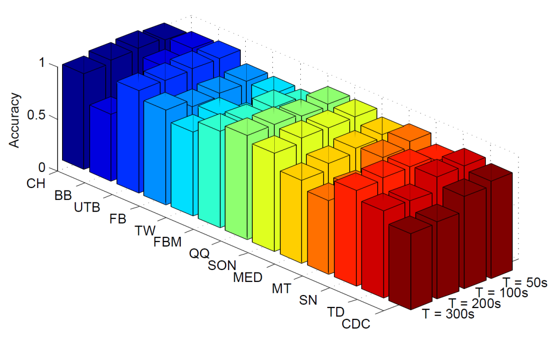

Fig. 6. Influence of monitoring time on true positive values. There is no clear patterns. Accuracy might slightly increase, remain the same or degrade with longer monitoring times.

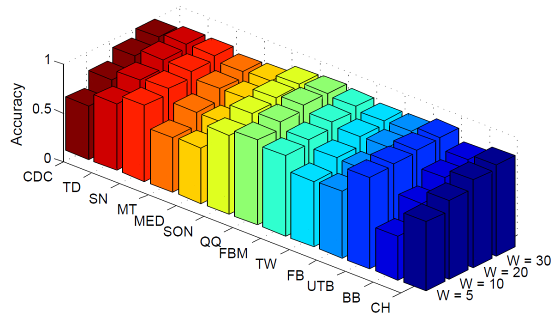

Fig. 7. Influence of window size on true positive values. While accuracy varies considerably among apps at small $W = 5$, at $W = 30$, true positive values for all classes are more than 85%. This implies that taking $W = 30$ can be considered as sufficient to achieve high classification accuracy.

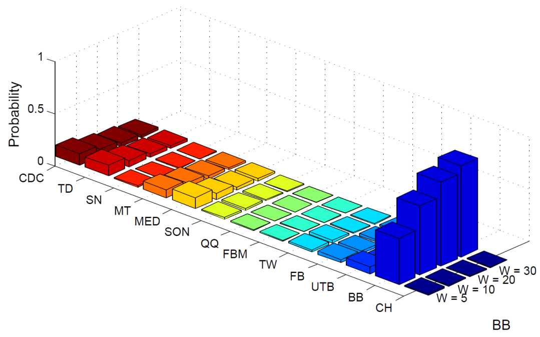

Fig. 8. Classification probability distributions for Boom Beach with varying window sizes. At low window sizes, Boom Beach is relatively often misclassified as CDC, Tinder, or Songza and only correctly classified less than 50% of the times.

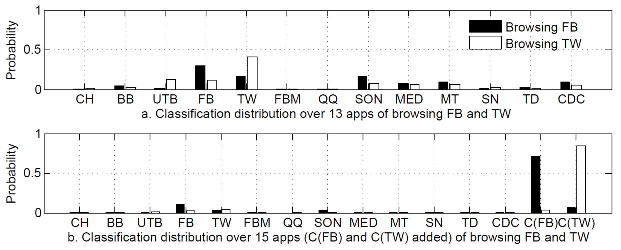

Fig. 9. Classifying using social networks on browsers vs. apps. Our classifier trained on app traffic (top figure) is able to detect Facebook activity through the browser with a success rate of close of 30%. For Twitter the positive value percentage is over 40%. This shows that the traffic generated through a service app is considerable but not completely different to the traffic of this service through the web-browser. Considering browser activity as independent apps (bottom figure) provides much better accuracy.

may be too many combinations of concurrent apps when the number of known apps are large. Secondly, new apps are appearing continuously, and older apps are becoming unpopular rendering them unimportant. Thirdly, any given attacker is likely only interested in a small subset of apps. As a result, the attacker would not be interested in maintaining and updating a very large database of apps, keeping up with traffic pattern changes and the resulting necessary updates to the classifier. Therefore, an effective and efficient inference method should be capable of identifying that a given traffic pattern does not belong to any of the apps under consideration.

Ideally, the testing data is not clearly classified to any particular application considered in the RF because this would lead to a false positive. Instead, different samples should be categorized by the classifier to belong to a number of different known apps. By having the tested data assigned to a wide range of apps, the attacker can derive that there are unknown apps or noise caused by concurrent apps.

We tested our classifier by running the following tests. For the detection of concurrent apps, we chose three combinations of apps, (1) music streaming app Songza and social app Facebook, (2) online chatting app Facebook Messenger and social app Snapchat, (3) unknown app Rdio and financial app Mint. These combinations include apps with large frame size, i.e. Songza and with medium frame size, i.e. Facebook, apps where both have small frame size, i.e. Snapchat and Facebook Messenger, and a combination of known app Mint and unknown app Rdio. For detecting unknown apps, we collected traffic from Rdio, Yelp, and LinkedIn, which were not included in our training samples. The probability distributions of all noisy cases are shown in Fig. 10.

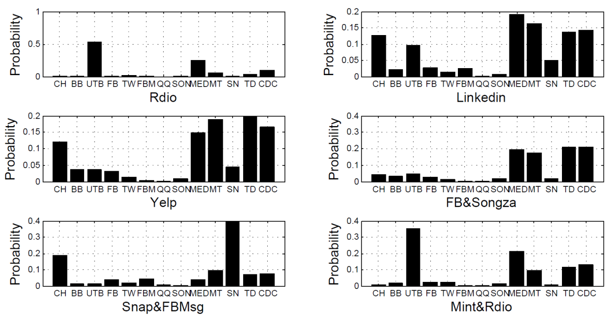

Fig. 10. Detection of Unknown and Concurrent Apps.

As we expected and hoped, the probability distribution of unknown apps are spread across many of the known apps. An exception is Rdio which is mostly identified as YouTube. This may be due to the fact that both Rdio and YouTube are streaming apps. However, in the cases of LinkedIn and Yelp, the highest probability for being classified as one of the known apps is less than 20%. A similar pattern can be observed in the case of concurrent apps.

V. Defense Mechanisms

The research on defense mechanisms against side-channel traffic analysis techniques is limited. While methods such as *traffic padding* and *traffic morphing* [23, 24] can battle these attacks they typically have high overhead problems even for computers and are not particularly suitable for less powerful mobile devices. Therefore, specific solutions are needed to effectively prevent traffic analysis attacks in mobile devices. One possible solution is to develop more intelligent scheduling algorithms for message transmission. This could be realized by delaying the transmission of packets from time-*in*sensitive apps. Since the inferring of users' activity is dependent on analyzing the statistical features of the generated traffic, intelligent scheduling can break easily identifiable patterns of traffic. In addition, as shown in the experiments, the detection of several concurrent apps is not as effective. Thus, by running multiple apps simultaneously, or combining their traffic with other apps more noise can be added to traffic.

VI. Conclusion

In this paper, we focus on the problem of side-channel information leaks in mobile systems. Here we show that these attacks, relying only on wireless traffic analysis, against mobile users are possible and we develop machine learning algorithms to verify the possibility of inferring users' online activities by analyzing packet-level traffic, even when the traffic is encrypted. These attacks are hard to defend against since they are not the result of security vulnerabilities in the OS or apps and the defense mechanisms are fairly limited and generally have a high cost. Furthermore, several aspects of mobile devices, like their reliance on native apps and limited multitasking, make it easier to perform traffic analysis attacks as they result in more identifiable traffic patterns. These attacks have a high accuracy and can be used in monitoring of a victim's behavior or to perform more targeted attacks based on their specific usage (such as detection of spoken words if a VoIP app is detected). A wide range of experiments are done to verify the feasibility and accuracy of these attacks and we believe that these attacks can be further used in different wireless networks including cellular networks and detection of new apps can be fairly easily added to the system.

References

[1] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[2] S. Chen, R. Wang, X. Wang, and K. Zhang, "Side-channel leaks in web applications: A reality today, a challenge tomorrow," in *Security and Privacy (SP), 2010 IEEE Symposium on*, May 2010, pp. 191–206.

[3] comScore Reports, "Smartphone Subscriber Market Share," https://www.comscore.com/Insights/Press-Releases /2014/6/comScore-Reports-April-2014-US-Smartphone-Subscriber-Market-Share, 2014.

[4] M. Conti, L. V. Mancini, R. Spolaor, and N. V. Verde, "Can't you hear me knocking: Identification of user actions on android apps via traffic analysis," in *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*. ACM, 2015, pp. 297–304.

[5] A. Dainotti, W. de Donato, A. Pescape, and P. Salvo Rossi, "Classification of network traffic via packet-level hidden markov models," in *Global Telecommunications Conference, 2008. IEEE GLOBECOM 2008. IEEE*, Nov 2008, pp. 1–5.

[6] T. G. Dietterich, "An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization," *Machine learning*, vol. 40, no. 2, pp. 139–157, 2000.

[7] T. M. T. Do, J. Blom, and D. Gatica-Perez, "Smartphone usage in the wild: A large-scale analysis of applications and context," in *Proc. of Multimodal Interfaces*, ser. ICMI. ACM, 2011, pp. 353–360.

[8] J. Du and Y.-C. Wu, "Distributed clock skew and offset estimation in wireless sensor networks: Asynchronous algorithm and convergence analysis," *Wireless Communications, IEEE Transactions on*, vol. 12, no. 11, pp. 5908–5917, November 2013.

[9] H. Falaki, R. Mahajan, S. Kandula, D. Lymberopoulos, R. Govindan, and D. Estrin, "Diversity in Smartphone Usage," in *Proc. of International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '10. New York, NY, USA: ACM, 2010, pp. 179–194.

[10] H. Huang, K. Chen, C. Ren, P. Liu, S. Zhu, and D. Wu, "Towards discovering and understanding unexpected hazards in tailoring antivirus software for android," in *Proc. of ACM Symp. on Information, Computer and Communications Security ICCS*. ACM, 2015, pp. 7–18.

[11] T. Jiang, H. J. Wang, and Y.-C. Hu, "Preserving location privacy in wireless lans," in *Proc. of Mobile systems, applications and services*. ACM, 2007, pp. 246–257.

[12] M. Liberatore and B. N. Levine, "Inferring the source of encrypted http connections," in *Proc. of Computer and communications security*. ACM, 2006, pp. 255–263.

[13] H. McCracken, "Whos winning, ios or android? all the numbers, all in one place," *Time (April 16, 2013)*, 2013.

[14] Y. Michalevsky, G. Nakibly, A. Schulman, and D. Boneh, "Powerspy: Location tracking using mobile device power analysis," *arXiv preprint arXiv:1502.03182*, 2015.

[15] A. W. Moore and D. Zuev, "Internet traffic classification using bayesian analysis techniques," in *Proc. of Measurement and Modeling of Computer Systems*, ser. SIGMETRICS. ACM, 2005, pp. 50–60.

[16] R. Ouyang, A.-S. Wong, and C.-T. Lea, "Received signal strength-based wireless localization via semidefinite programming: Noncooperative and cooperative schemes," *Vehicular Technology, IEEE Transactions on*, vol. 59, no. 3, pp. 1307–1318, March 2010.

[17] PewResearch Internet Project, "Fact Sheets Mobile Technology Fact Sheet," http://www.pewinternet.org/fact-sheets/mobile-technology-fact-sheet, 2014.

[18] W. Ren, L. Yu, L. Ma, and Y. Ren, "Rise: A reliable and secure scheme for wireless machine to machine communications," *Tsinghua Science and Technology*, vol. 18, no. 1, pp. 100–117, Feb 2013.

[19] T. Stöber, M. Frank, J. Schmitt, and I. Martinovic, "Who do you sync you are?: Smartphone fingerprinting via application behaviour," in *Proc. of Security and Privacy in Wireless and Mobile Networks*, ser. WiSec. ACM, 2013, pp. 7–12.

[20] Q. Sun, D. R. Simon, Y.-M. Wang, W. Russell, V. N. Padmanabhan, and L. Qiu, "Statistical identification of encrypted web browsing traffic," in *Security and Privacy, 2002. Proceedings. 2002 IEEE Symposium on*. IEEE, 2002, pp. 19–30.

[21] N. V. Verde, G. Ateniese, E. Gabrielli, L. V. Mancini, and A. Spognardi, "No nat'd user left behind: Fingerprinting users behind nat from netflow records alone," in *Distributed Computing Systems (ICDCS), 2014 IEEE 34th International Conference on*. IEEE, 2014, pp. 218–227.

[22] C. V. Wright, L. Ballard, S. E. Coull, F. Monrose, and G. M. Masson, "Spot me if you can: Uncovering spoken phrases in encrypted voip conversations," in *Security and Privacy, 2008. SP 2008. IEEE Symposium on*. IEEE, 2008, pp. 35–49.

[23] C. V. Wright, S. E. Coull, and F. Monrose, "Traffic morphing: An efficient defense against statistical traffic analysis," in *In Proc. of Network and Distributed Security Symposium*. IEEE, 2009, pp. 237–250.

[24] F. Zhang, W. He, Y. Chen, Z. Li, X. Wang, S. Chen, and X. Liu, "Thwarting wi-fi side-channel analysis through traffic demultiplexing," *IEEE Transactions on Wireless Communications*, vol. 13, no. 1, pp. 86–98, Jan 2014.

[25] F. Zhang, W. He, X. Liu, and P. G. Bridges, "Inferring users' online activities through traffic analysis," in *Proc. of Security and Privacy in Wireless and Mobile Networks*, ser. WiSec. ACM, 2011, pp. 59–70.

[26] N. Zhang, K. Yuan, M. Naveed, X. Zhou, and X. Wang, "Leave me alone: App–level protection against runtime information gathering on android," in *IEEE Symp. on Security & Privacy*. IEEE, 2015.