The Dark Side(-Channel) of Mobile Devices: A Survey on Network Traffic Analysis

Mauro Conti, QianQian Li, Alberto Maragno, and Riccardo Spolaor

Abstract—In recent years, mobile devices (e.g., smartphones and tablets) have met an increasing commercial success and have become a fundamental element of the everyday life for billions of people all around the world. Mobile devices are used not only for traditional communication activities (e.g., voice calls and messages) but also for more advanced tasks made possible by an enormous amount of multi-purpose applications (e.g., finance, gaming, and shopping). As a result, those devices generate a significant network traffic (a consistent part of overall Internet traffic). For this reason, the research community has been investigating security and privacy issues that are related to the network traffic generated by mobile devices, which could be analyzed to obtain information useful for a variety of goals (ranging from device security and network optimization, to fine grained user profiling).

In this paper, we review the works that contributed to the state of the art of network traffic analysis targeting mobile devices. In particular, we present a systematic classification of the works in the literature according to three criteria: (i) the goal of the analysis; (ii) the point where the network traffic is captured; and (iii) the targeted mobile platforms. In this survey, we consider points of capturing such as Wi-Fi Access Points, software simulation, inside physical devices or emulators. We also review and compare the different models and techniques used in the surveyed works to carry out their analyses, which could be ported to other emerging domains (e.g., Internet of Things and Software Defined Networking). We believe our survey will be a reference work for researchers and practitioners in this research field.

Index Terms—Internet traffic, machine learning, mobile device, network traffic analysis, smartphone, tablet computer.

I. INTRODUCTION

The last decade has been marked by the rise of mobile devices which are nowadays widely spread among people. The most diffused examples of such mobile devices are the smartphone and the tablet. When compared with traditional cell phones, smartphones and tablets (henceforth also referred as mobile devices) have an enormously increased computational power, more available memory, a larger display, and the Internet connectivity via both Wi-Fi and cellular networks. Moreover, such devices run mobile operating systems which are able to experience multimedia contents, as well as to run mobile applications (also called apps). Combined together, these elements enable both smartphones and tablets to have the same functionalities typically offered by laptops and desktop computers.

According to the statistics reported in [1], smartphone users were 25.3% of the global population in 2015, and this percentage is expected to grow till 37% in 2020. Similarly, the statistics about tablets reported in [2] indicate a global penetration of 13.8% in 2015, expected to reach 19.2% in 2020. The driving forces of this tremendous success are the ubiquitous Internet connectivity, thanks to the worldwide deployment of cellular and Wi-Fi networks, and a large number of apps available in the official (and unofficial) marketplaces. A mobile device typically hosts a lot of sensitive information about its owner, such as contacts, photos and videos, and GPS position. Such information has to be properly protected, especially when it is transmitted to remote services. Since an important fraction of the overall Internet traffic is due to mobile devices, it is not surprising that attackers and network traffic analysts have soon started to target them. For this reason, the research community investigates on network traffic analysis techniques to improve both security and privacy on mobile devices.

Network traffic analysis (henceforth simply referred as traffic analysis) is the branch of computer science that studies inferential methods which take the network traces of a group of devices (from a few to many thousands) as input, and give information about those devices, their users, their apps, or the traffic itself as output. Network traces can be captured at different layers (e.g., data-link layer, application layer), different points (e.g., within a Wi-Fi network, within the devices), and their content is often encrypted (making analysis even more challenging). Typically, researchers follow two different approaches for analyzing mobile network traffic: (i) taking pre-existent methods designed for traditional Internet traffic, and adapting them to the mobile scenario; or (ii) developing new methods tailored to mobile Internet traffic properties. It is worth to underline that this survey focuses on Internet traffic only. We do not consider other types of mobile traffic (e.g., Call Detail Records) or data transmission technologies (e.g., Bluetooth, Infrared).

Contributions — In this paper, we survey the state of the art of network traffic analysis on mobile devices, giving the following contributions:

- We categorize each work according to three criteria: 1) the goal of the analysis; 2) the point where the network traffic is captured (henceforth simply referred to as point of capturing); and 3) the targeted mobile platforms.
- Moreover, We provide further insights on the models and methods that can be used to perform traffic analysis targeting mobile devices.
II. Categorization of Work

In this section, we present an overview of the classification criteria we follow to categorize the works considered in our survey: the goal of the analysis performed on the mobile traffic (Section II-A); the point of capturing, which is where the mobile traffic is collected (Section II-B); and the targeted mobile platform (Section II-C). In Table I, we report the surveyed works according to these criteria. Moreover, for each work we indicate whether the proposed analyses are still applicable in case of traffic encryption via either SSL/TLS or IPsec (see Section IV for more details about how traffic encryption affects the analyses presented in the surveyed works). It is worth to notice that a few works (i.e., Wei et al. [4], and Tadrous and Sabharwal [5]) propose multiple traffic analysis techniques, each affected by traffic encryption in a different way.

A. Classification by Goal of the Analysis

The first classification takes into account the goal of the analysis performed on the captured mobile traffic. For each surveyed work, Table I provides this information in the Goal of the Analysis column. We survey more in detail the works according to this classification in Section III.

Overall, we are able to identify fourteen goals. In Figure 2, we depict such goals by their field of pertinence: apps, mobile users, and mobile devices. In what follows, we list and briefly describe each of the goals:

- **Ad fraud detection** (Ad Fraud in Table I): to detect ad fraud by a mobile app, i.e., to recognize if a mobile app is trying to trick the advertising business model (e.g., fabricating false user clicks on ads). This type of analysis is fundamental for ad providers, which can protect themselves from dishonest app developers trying to illicitly earn money. More details in Section III-A.
- **App identification** (App in Table I): to recognize the network traffic belonging to a specific mobile app. This type of analysis can help network administrators in resource planning and management, as well as in app-specific policy enforcement (e.g., forbidding a social network app within an enterprise network). Moreover, app identification can be employed to uncover the presence of sensitive apps (e.g., dating, health, religion) in the mobile device of a target user. See Section III-B.
- **Device positioning** (Positioning in Table I): to discover the geographical position of a mobile device. This type of analysis helps infer social status, interests, and habits of the owner of a mobile device. As a further step, the profiles of several mobile users can be aggregated for marketing, as well as sociological studies. See Section III-C.
- **Malware detection** (Malware in Table I): to detect whether a mobile app behaves maliciously (e.g., downloading and installing malicious code from the network). This type of analysis can be used to assess the security of an app submitted by a developer to a mobile marketplace. In such case, the result of the security tests decides whether the app can be released to the public. Moreover, malware detection algorithms can be embedded into anti-virus apps that mobile

A. Organization

The rest of the document is organized as follows. In Section II, we present the classifications adopted to report the works. In the following sections, we survey the works according to three criteria: in Section III the goal of the analysis performed on the mobile traffic; in Section IV the point of capturing; and, in Section V the targeted mobile platforms. In Section VI we review the models and methods applied in the surveyed works to perform traffic analysis targeting mobile devices. Finally, we conclude the paper in Section VII.
| Year | Paper | Goal of the Analysis | Point of Capturing | Targeted Mobile Platform | SSL/TLS | IPsec |
|------|-------|----------------------|-------------------|-------------------------|---------|-------|
| 2010 | Afanasiev et al. | Characterization, Usage | APs, Wired | Platform-independent | ✗ ✗ |
|      | Falaki et al. | Characterization, Usage | Devices | Android, Windows Mobile | ✓ ✓ |
|      | Hueste et al. | Positioning | Simulator | Platform-independent | ✓ ✓ |
|      | Maier et al. | Characterization, Usage | Wired | Platform-independent | ✗ ✗ |
|      | Shepard et al. | Characterization | Devices | iOS | ✓ ✗ |
| 2011 | Finamore et al. | Characterization, Usage | Wired | Platform-independent | ✗ ✗ |
|      | Gember et al. | Characterization, Usage | Devices | Android, Windows Mobile | ✓ ✓ |
|      | Lee et al. | Characterization, Usage | Wired | Platform-independent | ✗ ✗ |
|      | Maier et al. | Characterization | Devices | Android | ✓ ✓ |
|      | Shepard et al. | Characterization | Devices | iOS | ✓ ✗ |
| 2012 | Baghel et al. | Characterization | Wired | Android | ✓ ✓ |
|      | Chen et al. | Characterization | Wired | Platform-independent | ✗ ✗ |
|      | Ham et al. | Usage | Devices | Android | ✓ ✓ |
|      | Masa et al. | Trajectory | Monitors | Platform-independent | ✓ ✓ |
|      | Shabtai et al. | Malware | Devices | Android | ✓ ✓ |
|      | Stevens et al. | Ph Leakage | APs | Android | ✗ ✗ |
|      | Su et al. | Malware | Devices | Android | ✓ ✓ |
|      | Wei et al. | Characterization | Wired | Android, iOS | ✗ ✗ |
|      | Wei et al. | Characterization | Devices | Android | ✓ ✓ |
|      | Barbera et al. | Sociological | Monitors, Wired | Platform-independent | ✓ ✓ |
|      | Coull et al. | User Actions, USE | Devices | iOS | ✓ ✓ |
|      | Coull et al. | User Actions, OS | Devices | Android | ✓ ✓ |
|      | Lindorfer et al. | Characterization | Emulators | Android | ✗ ✗ |
|      | Shabtai et al. | Malware | Devices | Android | ✓ ✓ |
|      | Verde et al. | User Fingerprinting | Wired | Platform-independent | ✓ ✓ |
| 2013 | Chen et al. | Characterization | Wired | Android | ✓ ✓ |
|      | Fukuda et al. | Characterization, Usage | Devices | Android, iOS | ✓ ✓ |
|      | Le et al. | App, Ph Leakage | Devices | Android | ✓ ✓ |
|      | Park et al. | User Actions | Wired | Android | ✓ ✓ |
|      | Song et al. | Ph Leakage | Devices | Android | ✓ ✓ |
|      | Wang et al. | App | Monitors | iOS | ✓ ✓ |
|      | Yao et al. | App | Devices, Emulators | Android, iOS, Symbian | ✗ ✗ |
|      | Zaman et al. | Malware | Devices | Android | ✓ ✓ |
|      | Alan et al. | App | Devices | Android | ✓ ✓ |
|      | Comit et al. | User Actions | Wired | Android | ✓ ✓ |
|      | Mongkolratanat et al. | Maschine | Devices, Emulators | Android | ✓ ✓ |
| 2015 | Narin et al. | Malware | Devices, Emulators | Android | ✓ ✓ |
|      | Nayam et al. | Characterization | Wired | Android, iOS | ✗ ✗ |
|      | Ren et al. | Ph Leakage | Wired | Android, iOS, Windows Phone | ✗ ✗ |
|      | Ruiling et al. | OS | Monitors | Android, iOS, Windows Phone, Symbian | ✓ ✓ |
|      | Safarius et al. | User Actions | Devices | Android | ✓ ✓ |
|      | Spreiter et al. | Website Fingerprinting | Devices | Android | ✓ ✓ |
|      | Tairos et al. | Characterization | Devices | Android, iOS | ✓ ✓ |
|      | Vanryck et al. | Ph Leakage, User Fingerprinting | Wired | Android | ✗ ✗ |
|      | Wang et al. | Malware | Devices | Android | ✓ ✓ |
|      | Arora et al. | Malware | Devices | Android | ✓ ✓ |
|      | Continella et al. | Ph Leakage | Wired | Android | ✗ ✗ |
|      | Espada et al. | Characterization | Devices | Android | ✓ ✓ |
|      | Matik et al. | OS | Devices | Android, iOS, Windows Phone | ✓ ✓ |
|      | Taylor et al. | App | Wired | Android | ✓ ✗ |
|      | Wei et al. | Characterization, Usage | Wired | Platform-independent | ✗ ✗ |
users can use to check whether an installed app is malicious. See Section III-D

- Operating system identification (OS in Table I): to discover the operating system of a mobile device. This type of analysis is usually a preliminary phase for more advanced attacks against mobile devices: the adversary tries to infer the operating system of the target mobile device in order to subsequently exploit an ad-hoc vulnerability for that specific OS. Moreover, operating system identification carried out on a large mobile user population can be a starting point for other types of analysis not directly related to computer science (e.g., sociological studies). See Section III-E

- PII leakage detection (PII Leakage in Table I): to detect and/or prevent the leakage of a mobile user’s Personal Identifiable Information (PII). This type of analysis can be employed to assess the behavior of a mobile app from a privacy point of view, by checking which PII it actually discloses to remote hosts. Detecting PII leakage is also the first step to prevent such problem, since it is then possible to block network transmissions carrying PII, or replace sensitive information with bogus data. See Section III-F

- Sociological inference (Sociological in Table I): to infer some kind of sociological information about mobile users (e.g., language, religion, health condition, sexual preference, wealth), from one or more properties related to their mobile devices (e.g., list of installed apps, associated Wi-Fi networks). See Section III-G

- Tethering detection (Tethering in Table I): to detect if a mobile device is tethering, i.e., it is sharing its Internet connectivity with other devices, for which it acts as an access point. Tethering constitutes a problem for cellular network providers, since it significantly increases the volume of network traffic generated by a single client. Such providers are therefore interested in tethering detection techniques that can be used to prevent their customers from sharing their Internet connectivity, or simply require them to pay an extra fee to do that. See Section III-H

- Traffic characterization (Characterization in Table I): to infer the network properties of mobile traffic. The knowledge of such properties is crucial to effectively deploy and configure the resources in cellular networks, as well as in Wi-Fi networks serving mobile devices. See Section III-I

- Trajectory estimation (Trajectory in Table I): to estimate the trajectory (i.e., the movements) of a mobile device within a geographical area. This type of analysis can help study the interests and social habits of a mobile user. Moreover, trajectory estimation can aid road traffic prediction along urban streets, by leveraging the most frequent trajectories followed by the citizens that move along the city. See Section III-J

- Usage study (Usage in Table I): to infer the usage habits of mobile users (e.g., which are the most frequently used apps). As an example, the knowledge of the places where mobile devices are mostly used can drive the deployment of cellular stations and Wi-Fi hotspots. See Section III-K

- User action identification (User Actions in Table I): to identify a specific action that a mobile user performed on her mobile device (e.g., uploading a photo on Instagram), or to infer some information about that specific action (e.g., the length of a mobile user’s message sent through an instant messaging app). Researchers can employ such analysis to discover the identity behind an anonymous social network profile. This can be accomplished by verifying if there is a match between the events reported on that profile’s page, and the actions a suspect performed while using the mobile app of that social network. Alternatively, it is possible to build behavioral profiles of mobile users, which are useful for user reconnaissance within networks and, in aggregated form, for marketing studies. See Section III-L

- User fingerprinting (User Fingerprinting in Table I): to detect the traffic belonging to a specific mobile user. This type of analysis can be employed to trace a mobile user, by approximating her position with the location of the Wi-Fi hotspot or cellular station to which her mobile device
is connected. From this information, it is then possible to build a behavioral profile of that mobile user. Alternatively, it is possible to examine a mobile traffic dataset in order to extract and group together the network traces generated by a specific mobile user. Such data can be subsequently used for other types of traffic analysis targeting that user. See Section III-M.

- Website fingerprinting (Website Fingerprinting in Table I): to infer which websites and/or webpages are visited by a mobile user while navigating via the web browser of her mobile device. Similarly to sociological inference, this type of analysis can reveal interests, social habits, religious belief, as well as sexual and political orientations of a mobile user. See Section III-L.

B. Classification by Point of Capturing

The second classification considers where and how the mobile traffic is captured. For each surveyed work, Table I provides this information in the Point of Capturing column. It is worth to notice that: (i) we are focusing on the (hardware and/or software) equipment that captures the traffic; and (ii) we report the point of capturing only for those datasets for which the authors give enough details about the collection process. We survey more in detail the works according to this classification in Section IV.

Overall, we identify six different points of capturing:

- Within one or more mobile devices, i.e., client-side (Devices in Table I). This type of point of capturing is particularly useful if we want to target a specific mobile app (e.g., Facebook), or a particular network interface (e.g., cellular). We specify that this category also includes the case of a network traffic logger installed within either: (i) a mobile device emulator; and (ii) a machine to which the mobile traffic is mirrored using a remote virtual network interface. More details in Section IV-A.

- At one or more wired network equipments (Wired in Table I). The size of the population of monitored mobile devices varies according to the type of considered network equipments: thousands of mobile users in the case of edge routers (i.e., routers connecting customers to the ISP’s backbone) and Internet gateways; from tens to a few hundreds in the case of VPN servers and forwarding servers (i.e., traditional desktop computer put in the middle of a wired link and set up to log all traffic traversing it). See Section IV-B.

- At one or more access points of a Wi-Fi network (APs in Table I). This type of point of capturing allows the number of monitored mobile devices to vary from tens to a few thousands, and it is suitable to capture the traffic of mobile devices while their users are performing network-intensive tasks (e.g., watching streaming videos, updating apps). See Section IV-C.

- At one or more Wi-Fi monitors (Monitors in Table I). Researchers usually employ this type of capturing devices to focus the network traffic collection process on a specific geographical area of interest (e.g., a train station). Such approach is often the only viable solution whether it is not possible to directly access a target mobile device, or the network to which it is connected. See Section IV-D.

- At one or more machines running virtual mobile devices, i.e., emulators (Emulators in Table I). The possibility to run multiple virtual mobile devices in parallel and control them via automated tools enables a large-scale network traffic collection that would be more expensive if conducted on real mobile devices. It is important to highlight that the traffic logging is performed by the host machines or their virtualization managers. We do not consider the case in which the traffic logging takes place within the emulated mobile devices (such case is covered by the Devices category). See Section IV-E.

- At one or more virtual capturing points within a simulated environment generated and managed by a software program (Simulator in Table I). This point of capturing can help study particular deployments of mobile devices that are not observable in a real-world scenario because of technical, economical, or legal constraints. See Section IV-F.

C. Classification by Targeted Mobile Platform

The third classification considers the mobile platforms that are targeted by the traffic analysis. For each surveyed work, Table I provides this information in the Targeted Mobile Platform column. It is worth to specify that we classify a work as platform-independent if its authors do not provide information about the targeted mobile platforms, or such information is not relevant to the analysis they perform on the mobile traffic. We survey more in detail the works according to this classification in Section V.

Overall, we find four distinct mobile platforms: Android, Google’s open-source mobile operating system (we discuss it in Section V-A); iOS, the operating system of Apple’s mobile devices (Section V-B); Symbian, the first released modern mobile operating system (Section V-C); and Windows Mobile/Phone, the mobile counterpart of Microsoft’s desktop operating system (Section V-D).

III. GOALS OF TRAFFIC ANALYSIS TARGETING MOBILE DEVICES

In this section, we survey the works according to the goal of the analysis that is performed on the mobile traffic. Table II summarizes the goals of the surveyed works. As shown in Figure 3, the most frequently pursued goal is traffic characterization (eighteen works), followed by usage study (ten works), app identification (nine works), malware detection and PII leakage detection (eight works each), user action identification (five works), operating system identification (four works), and user fingerprinting (two works). Each of the following goals counts one work only: ad fraud detection, device positioning, sociological inference, tethering detection, trajectory estimation, and website fingerprinting. As shown in Table II, twelve works pursue two goals, and one work even three. In the following sections, we present the goal(s) and achieved results for each surveyed work. We also discuss whether the proposed analysis works on encrypted network traffic (e.g., IPsec, SSL/TLS).
TABLE II
THE SURVEYED WORKS BY GOAL OF THE ANALYSIS PERFORMED ON THE CAPTURED MOBILE TRAFFIC.

| Year | Paper | Ad Fraud Detection | App Identification | Device Positioning | Malware Detection | Operating System Identification | Traffic Characterization | Traffic Estimation | Usage Study | User Action Identification | User Fingerprinting | Website Fingerprinting |
|------|-------|--------------------|--------------------|-------------------|------------------|---------------------------------|------------------------|-----------------|------------|--------------------------|---------------------|------------------------|
| 2010 | Afanasyev et al. [22] | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
|       | Fu et al. [23] | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
|       | Naryan et al. [24] | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| 2011 | Pinamone et al. [11] | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
|       | Gember et al. [12] | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
|       | Lee et al. [13] | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
|       | Rao et al. [14] | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
|       | Baghel et al. [15] | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
|       | Chen et al. [16] | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
|       | Kim et al. [17] | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
|       | Mina et al. [18] | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| 2012 | Shabtai et al. [19] | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
|       | Stevens et al. [20] | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
|       | Su et al. [21] | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
|       | Wei et al. [22] | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
|       | Stavrou et al. [23] | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
|       | Watkinson et al. [24] | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| 2013 | Sauter et al. [25] | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
|       | Konidri et al. [26] | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
|       | Crussell et al. [27] | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
|       | Lai et al. [28] | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
|       | Fukuda et al. [29] | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
|       | Le et al. [30] | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
|       | Park et al. [31] | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
|       | Song et al. [32] | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
|       | Wang et al. [33] | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
|       | Yao et al. [34] | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
|       | Zaman et al. [35] | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| 2014 | Chen et al. [36] | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
|       | Crussell et al. [37] | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
|       | Konidri et al. [38] | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
|       | Sauter et al. [39] | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
|       | Verweij et al. [40] | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
|       | Konidri et al. [41] | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
|       | Fukuda et al. [42] | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
|       | Le et al. [43] | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
|       | Park et al. [44] | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
|       | Song et al. [45] | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
|       | Wang et al. [46] | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
|       | Yao et al. [47] | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
|       | Zaman et al. [48] | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| 2015 | Zaman et al. [49] | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
|       | Konidri et al. [50] | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
|       | Konidri et al. [51] | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
|       | Konidri et al. [52] | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
|       | Konidri et al. [53] | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
|       | Konidri et al. [54] | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
|       | Konidri et al. [55] | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
|       | Konidri et al. [56] | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
|       | Konidri et al. [57] | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
|       | Konidri et al. [58] | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
|       | Konidri et al. [59] | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |

A. Ad Fraud Detection

Mobile apps can be partitioned into two macro categories: paid and free apps. To cover the cost of developing and maintaining a free app, developers usually rely on advertisement, which applies the following business model:

- The ad provider yields a library that can be embedded in the app. Such library fetches ad contents and displays them on the app’s user interface.
- The ad provider pays according to the amount of times the ads are displayed to the user (impressions) and/or clicked by the user (clicks).

We define as ad fraud detection the analysis of network traffic in order to uncover apps that trick the business model described above and let their developers illicitly earn money. Crussell et al. in [30] focus on the Android platform. They identify two fraudulent app behaviors:

- To request ads while the app is running in background. This generates impressions without actually displaying ads to the user.
- To click on ads without any user interaction, which is achievable in the following ways: (i) the app can trick the ad library by simulating a user click on the ad with a touch event; and (ii) the app extracts the click URL from the ad request (i.e., the web page that will be opened when the user clicks on the ad), then makes an HTTP request to the click URL to simulate a user click.

The authors propose MAdFraud, a tool being able to automatically run Android apps in emulators and analyze their application-layer traffic in order to expose ad fraud. The system is employed to analyze 130,339 Android apps crawled from nineteen different marketplaces, and 35,087 Android apps that probably contain malware (provided by an unspecified security company). The authors report that 30% of apps generate fake impressions (i.e., they request to display an ad while running in the background), while 27 apps generate fake clicks (i.e., they contact a click URL without any user interaction). Unfortunately, MAdFraud cannot process encrypted traffic, since it relies on the HTTP and DNS data generated by apps. However, the authors’ analysis covers most of the available ad libraries. This means that such libraries do not
usually employ any form of encryption for their data transfers, and simply rely on plain HTTP.

B. App Identification

The Internet connectivity and multi-purpose apps are two key aspects of the success and widespread adoption of mobile devices. Most of the apps can send and receive data through the network interfaces of mobile devices (i.e., Wi-Fi and cellular), and often this capability is mandatory for apps to properly work.

The network traffic patterns related to an app (or type of app) constitute a behavioral network fingerprint which can be recognized in unseen network traces. We refer to this type of analysis as app identification. It is worth to notice that this approach also takes into consideration the network traffic of an app that is not directly related to user actions (e.g., the data exchanged because of background activities).

App identification has several applications as well as privacy implications:

- The knowledge of the apps used by the clients of a network can help the administrators to tune the network equipments and parameters in order to deliver the best achievable Quality of Service (QoS).
- In an enterprise network where the use of particular apps is not allowed (e.g., Facebook, Twitter), app detection can help the administrators enforce such policy by blocking traffic belonging to the forbidden apps.
- It is possible to target a high profile user and discover whether she uses privacy-sensitive (e.g., health, dating) apps.

In Table III we report the surveyed works that deal with app identification [13, 25, 26, 36, 40, 41, 43, 45, 58]. The number of apps selected for profiling and fingerprinting varies greatly, from less than ten to many thousands.

Despite the core topic of the work by Lee et al. in [13] is a comparison between smartphone traffic and traditional Internet traffic, the authors also perform app identification targeting Android and iOS platforms. Indeed, they select the top 50 apps of both Apple App Store and Google Play Store, generate their payload signatures, and use such signatures to recognize the traffic generated by such apps in network traces. The classifier matches only 15.37% of flows with one of the considered apps, testifying that smartphones users install a variety of apps on their devices. Unfortunately, this approach is based on payload signatures, thus it cannot deal with the apps that encrypt their network traffic.

Qazi et al. in [25] present the Atlas framework, which incorporates application identification into Software-Defined Networking (SDN). Prototyped on HP Labs wireless network, Atlas is tested for Android app identification. The authors select the 40 most popular Android apps and collect over 100,000 network flows which are used to train a machine learning classifier that reaches 94% F-measure on average. Since it requires to inspect transport-layer information, Atlas cannot process network traffic protected by IPsec.

Rao et al. in [26] present Meddle, a cross-platform system for collecting and analyzing the network traffic of mobile devices. The idea is to leverage VPN tunnels (which are natively supported by modern mobile OS) to redirect the network traffic of the target mobile devices to the Meddle proxy server, where software middleboxes are responsible for traffic processing and analysis. Thanks to its man-in-the-middle approach, Meddle can inspect the network traffic protected by SSL/TLS, but cannot deal with data transmissions encrypted via IPsec. The authors employ Meddle for app identification (and also PII leakage detection, see Section III-F for details) based on the Host and User-Agent fields of HTTP messages. On a dataset for which ground truth is available, the identification rate is 64% for Android apps, and 89% for iOS ones. On a real-world dataset with no ground truth and no SSL decryption (because of privacy reasons), over 92% of the flows, as well as 40% of the SSL ones, are mapped to the considered Android and iOS apps.

Le et al. in [36] propose AntMonitor, a system for collecting and analyzing network traffic from Android devices. Among other types of analysis, AntMonitor can perform app identification. To evaluate the performance of their solution, the authors select 70 Android apps and leverage a machine learning classifier which takes into account 84 network-level features. The system achieves 70.1% F-measure. Among the considered features, there are the flags of TCP segments, which are hidden if IPsec is employed, thus the proposed framework does not work if network-layer encryption is in place.

The app identification framework proposed by Wang et al. in [40] is based on extracting side-channel information from Wi-Fi traffic belonging to a target mobile device. The authors depict a passive adversary as follows: (i) she is able to sniff the traffic on the same WLAN channel as the access point to which the target device is connected; (ii) leveraging the MAC address of the target device, she can elicit its traffic from the collected network traces; and (iii) she cannot break the encryption scheme of the sniffed traffic (i.e., the app identification can also target secure WLANs). To evaluate their solution, the authors choose the iOS platform, considering thirteen popular apps from a wide range of different app categories. The reached accuracy is more than 90%.

Yao et al. in [41] present SAMPLES (Self Adaptive Mining of Persistent LEXical Snippets), an app identification framework that leverages the occurrences of app identifiers within HTTP headers (thus it cannot handle network traffic protected by IPsec or SSL/TLS). SAMPLES models such occurrences.

| Year | Paper | Number of Targeted Apps |
|------|-------|-------------------------|
|      |       | Android | iOS | Symbian |
| 2011 | Lee et al. [13] | 94 | None | None |
| 2013 | Qazi et al. [25] | 40 | None | None |
| 2015 | Le et al. [36] | 832 | 209 | None |
| 2016 | Alan et al. [43] | 1,505 | None | None |
| 2017 | Taylor et al. [58] | 110 | None | None |
into generalized conjunctive rules, which are used to identify the app that generated a given network flow. To evaluate their system, the authors consider over 700,000 apps from Google Play Store, Apple App Store and Nokia OVI Store (details are given in Table III). The proposed solution identifies over 90% of these apps with an average accuracy of 99%. Surprisingly, these results are obtained by training SAMPLES with less than 2% of the apps for each of the three considered marketplaces. Moreover, SAMPLES outperforms the solutions proposed by Tongaonkar et al. in [60] and Xu et al. in [61].

Alan and Kaur in [43] investigate the feasibility of identifying Android apps from their launch-time network traffic by only leveraging the information available in TCP/IP headers. The authors collect the launch-time traffic of 1,595 apps, and use it to train and evaluate three machine learning classifiers, achieving a maximum accuracy of 87% (more details on the classifiers are given in Section VI-C2). All the proposed classifiers can handle network traffic encrypted via SSL/TLS, and two of them also via IPsec.

Mongkolruksamee et al. in [55] (which is an extended version of a previous work by the same authors [62]) leverage machine learning to build an app identification system for Android apps. The authors leverage graphlet- and histogram-based features, and employ a random forest classifier (more details in Section VI-C2). The proposed system achieves 0.96 F-measure but it is evaluated on five Android apps only. However, it requires to access TCP and UDP headers, which is infeasible for apps that employ IPsec to encrypt their network traffic. It is worth to notice that despite this work focuses on 3G traffic, the actual capturing of the network traffic is performed within a mobile device via tcpdump (more details in Section IV-A).

Taylor et al. in [58] (which is an extension of a previous work by the same authors [63]) propose AppScanner, an app identification system based on machine learning (more details in Section VI-C2). The authors profile 110 popular Android apps crawled from Google Play Store and re-identify them in real-time with up to 96% accuracy. Moreover, the authors study how the classification performance is affected by varying the duration of the network traffic capturing, the mobile device that generates the collected data, and the version of the fingerprinted apps. AppScanner leverages the information within IP and TCP headers, thus being able to process the network traffic encrypted via SSL/TLS, but not the one protected by IPsec.

C. Device Positioning

The set of places frequently visited by a person tells a lot about her social status, interests, and habits. Such information can be exploited for commercial purposes (e.g., targeted advertisement), as well as intelligence activities (e.g., police investigations). Since most of the people own a mobile device and keep it with them all day long, locating the smartphone/tablet of a target user becomes a simple yet effective way to know her position. Moreover, it is possible to build a profile of the subject by aggregating multiple position detections.

We define as device positioning the inference of the geographical position of a mobile device by analyzing the network traffic it generates. In this section, we survey the works that propose this type of traffic analysis. We point out that we do not consider the works in which: (i) the mobile traffic is analyzed to detect the leakage of GPS coordinates (we consider this kind of works in Section III-F); (ii) the movements (not the position) of a mobile device are reconstructed from its network traffic (we review this kind of works in Section III-J); and (iii) the analysis performed on the network traffic is device-agnostic, i.e., it does not take into account the fact that the target devices are smartphones/tablets (this kind of works is excluded since it is too generic).

Husted and Myers in [8] investigate whether a malnet (i.e., a colluding network of malicious Wi-Fi devices) can successfully determine the location of a mobile device. Each malicious node looks for probe requests carrying the MAC address of the target mobile device, and uploads its findings to a central server where the data coming from all nodes is used for trilateration. Through a software simulation of a metropolitan population of users equipped with 802.11g mobile devices, the authors show that 10% of tracking population is sufficient to track the position of the remaining users. Besides, the tracking benefits from extending the broadcasting range of the mobile devices. This suggests that the adoption of newer 802.11 standards can make it feasible to build a geolocating malnet.

D. Malware Detection

As happened for personal computers, the success and widespread adoption of mobile devices have attracted the interest of malware developers. Mobile devices, and particularly smartphones, are an ideal target for attackers since: (i) they are ubiquitous, i.e., the population of potential targets is large; (ii) they host sensitive information about their owners (e.g., identity, contacts, GPS position); and (iii) they have networking capabilities and they are usually connected to the Internet.

We define as malware detection the attempt to understand if a mobile app is malicious through the analysis of the network traffic it generates. In this section, we survey the works that study the properties of the network traffic generated by malicious apps, because in such case the analysis is more related to traffic characterization (see Section III-I). From the surveyed works, we elicited that there are three kinds of actor which can be interested in malware detection: (i) an app marketplace [21]; (ii) a security company [22, 42, 46, 53, 54]; or (iii) a mobile user [19, 82].

Shabtai et al. in [19] present an anomaly-based malware detection app for Android devices. The detection model is based on machine learning. The proposed app monitors several aspects of the device (e.g., memory, network, power) and extracts different features, some of which are related to network traffic (e.g., the number of received packets). A properly trained classifier is then employed to check whether an installed app is malicious. The proposed solution is evaluated using 40 benign and 4 malicious Android apps. The authors consider different
classifiers (e.g., decision tree, Bayesian networks), as well as different metrics for feature selection (e.g., Fisher score, information gain). Moreover, the authors investigate how the detection accuracy is affected when: (i) the testing apps are not used in the training phase; and (ii) training and testing are performed on different devices. Overall, the proposed app achieves high accuracy, despite causing a limited performance overhead on the device.

Su et al. in [21] propose a framework that allows an Android marketplace to detect whether a new app submitted by a mobile developer is malicious. The system consists of servers, where developers can upload their new apps for verification, and physical Android devices, on which apps are actually executed while monitoring their system calls and network traffic. The gathered information is sent to a central server, which classifies each app as safe or malicious according to the response of two classifiers based on system call statistics and network traffic features, respectively. The network traffic classifier is trained with data from 49 malicious apps (from 22 malware families) and 60 safe apps, and tested with data from 50 malicious apps (from 22 malware families) and 70 safe apps (from eleven app categories). The best performing implementation of the classifier (i.e., random forest) reaches 96.7% accuracy. It is worth to notice that the classifier cannot process network traffic encrypted via IPsec, since one of the features it leverages is the average TCP session duration, which is not computable without accessing TCP headers.

Wei et al. in [22] present a framework for Android malware detection. Using network traffic generated by malicious Android apps, the system is trained in order to learn the network behavior of Android malware with regard to the resolution of domain names. After that, it can be employed to automatically analyze the DNS traffic of a given app and state if it is safe or malicious. The authors evaluate their solution using malicious apps from a public dataset of Android malware and benign apps from the official Android marketplace. The proposed classifier reaches nearly 1.0 accuracy, precision, and recall. A weakness of this framework is that it requires the access to the DNS traffic of apps, which can be hidden by IPsec or SSL/TLS encryption.

The work by Zaman et al. [42] stems from the observation that malicious apps usually send the user’s sensitive information to malicious remote hosts. The idea is to log all communications with remote hosts for each app installed on the mobile device. Leveraging a list of known malicious domains, it is possible to label the apps that contacted them as malware. This approach requires to inspect the URLs within HTTP messages, therefore it is not applicable to encrypted network traffic. The authors evaluate their solution on two Android malware samples: DroidKungFu and AnserverBot. The proposed technique only detects the former.

Narudin et al. in [66] investigate whether an anomaly-based IDS can successfully detect malicious Android apps by relying on traffic analysis. To build a comprehensive dataset of network traces, the authors run benign apps on a physical Android device and malicious apps on dynamic analysis platforms available online. The collected network traffic is then sent to a central server, where several machine learning classifiers (e.g., random forest, multi-layer perceptron) are trained and evaluated. All such classifiers reach over 90% true-positive rate on the network traces generated by the benign apps and several malicious apps provided by the Android Malware Genome Project, and over 73% on the network traces generated by the benign apps and 30 newly (in 2013) appeared malicious apps. The classifiers cannot process encrypted traffic, since they need to inspect HTTP messages.

Wang et al. in [53] present TrafficAV, an Android malware detection system based on machine learning. The proposed framework offers two distinct detection models, which rely on TCP- and HTTP-related network features, respectively. We give more details about considered features and classifiers in Section VI-C4. While model that rely on HTTP-related features cannot work on encrypted traffic since it requires DPI, both proposed models cannot cope with apps that employ IPsec. Evaluated on the network traffic of 8,312 benign apps and 5,560 malware samples, TrafficAV achieves a detection rate of 98.16% for the TCP-based model, and 99.65% for the HTTP-based model.

Arora and Peddoju in [54] (which is an extension and refinement of a previous work by the same authors [64]) also apply machine learning to detect Android malware. They collect the network traffic of malware samples from eleven families, extract 22 network-layer features (e.g., average time interval between received packets, per-flow sent bytes), and train a naive Bayes classifier. Evaluated on the network traffic of malware samples from six families (different from the ones used for training), the classifier reaches a detection rate of 87.25%. Moreover, the authors present a feature selection algorithm that reduces the number of features to be used, while limiting the drop in detection accuracy. The initial set of features is reduced from 22 to 9 elements, and the classifier achieves a detection rate of 83.3%. The proposed framework is encryption-agnostic, although the authors report that encryption may be a possible solution to evade detection. More details about the features and the algorithm to select them are provided in Section VI-C4.

Shabtai et al. in [32] design an anomaly-based malware detection app for Android devices. Such app analyzes the network behavior of the apps installed on the device in order to identify self-updating malware (i.e., benign apps that after being installed on the device, they download a malicious payload from the Internet) and popular apps republished with additional malicious code. The idea is to model the normal network behavior of each installed app as a set of traffic patterns, and subsequently detect any deviation from those patterns. The system is evaluated on several benign apps, ten self-updating malicious apps developed by the authors, and the infected version of five of the chosen benign apps. The authors claim that their method achieves a true-positive rate over 83%, and a false-positive rate below 12%. Moreover, the system works even with apps that encrypt their network traffic, since it needs to know only their amount of transmitted/received bytes and its percent out of the total device traffic.
E. Operating System Identification

We define as operating system identification the attempt to discover the operating system of a mobile device by analyzing its network traffic. This type of analysis has several applications:

- An adversary can identify the operating system of a target mobile device, and tailor her subsequent attack to that OS (e.g., by choosing a proper security exploit). In such case, the operating system identification is a preparatory task for more advanced and focused attacks. Moreover, the overall attack strategy can be more effective if the adversary is able to infer not only the operating system of the target mobile device, but also the version of that OS.

- It is possible to expose the adoption of mobile operating systems among a crowd of people. This can be a starting point for marketing, as well as sociological studies (we consider the latter in Section III-G).

In this section, we survey three works [28], [29], [49], [57]. Table IV reports the mobile operating systems they consider.

Chen et al. in [28] develop a probabilistic classifier that leverages the information available in IP and TCP headers, therefore it works unless IPsec is employed to hide the content of IP packets. Such classifier is evaluated using network traces captured at a Wi-Fi access point to which Android and iOS mobile devices, as well as Windows laptops are connected. The results show that the classifier reaches 1.0 precision and recall for iOS identification, and 1.0 precision and 0.8 recall for Android identification.

Coull and Dyer in [29] target iMessage, Apple’s instant messaging service. They leverage the sizes of the encrypted packets exchanged between the target user and Apple’s servers, in order to determine if she is using iMessage on iOS or OS X. The proposed classifier needs to observe only five packets to identify the OS with 100% accuracy.

The work by Ruffing et al. [49] stems from the observation that the timing of the network traffic generated by a mobile device depends on its operating system. The idea is to analyze the frequency spectrum of packet timing in order to identify the frequency components that are related to OS features, and filter out the ones that bring noise. Since this approach does not require to inspect the content of packets, it can be successfully applied even if encryption is in place. The authors evaluate their solution using network traffic captured from smartphones running the following operating systems: Android, iOS, Windows Phone, and Symbian. On 30-second-long traces, the proposed framework achieves around 70% success rate (i.e., the number of correct OS identifications over the total number of attempts). On traces lasting five minutes or more, the success rate is around 90%. Moreover, in case of heavy multitasking, only 30 seconds of traffic are sufficient to achieve 100% success rate. The authors also evaluate whether their approach is suitable to discriminate different versions of the same OS, and they choose Android and iOS for such analysis. On fifteen-minutes-long traces from the Skype app, the success rate is 98% and the misclassification rate is below 10%. On traces of the same length but generated by the YouTube app, the success rate is around 50%.

Malik et al. in [57] present a framework that exploits the inter-packet time of packets coming from a target mobile device in order to infer its operating system. In particular, the authors focus on two types of packet: (i) the response to an ICMP packet sent to the target mobile device (active measurement); and (ii) an IP packet related to a video stream involving the target mobile device (passive measurement). In both cases, the proposed solution effectively discriminates among three mobile operating systems, namely Android, iOS, and Windows Phone. Moreover, such approach is not hindered by traffic encryption, since it exploits the timing of packets. However, we must point out that the authors’ testbed includes only three devices, one for each of the considered mobile operating systems. Therefore, it is not clear how the model of the device and the version of the operating system affect the OS identification accuracy.

F. PII Leakage Detection

As we introduced in Section I a mobile device is a source of sensitive information about its owner (e.g., phone number, contacts, photos, videos, GPS position). In addition, apps often require to access such information to deliver their services. As an example, an instant messaging app (e.g., WhatsApp, Telegram, WeChat) requires to access the contacts saved in the device’s address book. As another example, a social network app (e.g., Facebook, Instagram) requires to inspect the device’s memory to find photos.

To disclose sensitive information to a remote host, an app must be authorized to: (i) access some kind of sensitive information (e.g., the GPS position); and (ii) connect to the Internet. The disclosure can be either allowed or illicit, depending on the level of sensitivity of the disclosed information, the reason why the app transmits such information to a remote host, and whether the user is aware of this transmission of sensitive data.

In this section, we focus on personal identifiable information (PII), which is information that can be used to identify, locate, or contact an individual. In the domain of mobile devices, there are four types of PII:

- Information related to mobile devices, such as IMEI (International Mobile Equipment Identity, a unique identifier associated to each mobile device), Android Device ID (an identifier randomly generated on the first boot of Android device), and MAC address (a unique identifier assigned to each network interface).

- Information related to SIM cards, such as IMSI (International Mobile Subscriber Identity, a unique identifier assigned to each subscriber of a cellular service), and SIM Serial ID (the identifier assigned to each SIM card).

- Information related to users, such as name, gender, date of birth, address, phone number, and email.

- Information about user’s location, such as GPS position and ZIP code.

We define as PII leakage detection the analysis of the network traffic of a mobile device in order to detect the leakage of user’s PII. Once a PII leakage is detected, it is possible to apply suitable countermeasures, such as blocking the network flows carrying the PII, or substituting the sensitive information
with bogus data. The latter approach is a good solution for mobile users who want to protect their privacy while being able to enjoy the functionalities of the apps.

In Table IV we present the surveyed works that deal with PII leakage detection [29], [24], [26], [53], [59], [49], [52], [65]. For each work, we summarize the targeted mobile platforms, and whether the PII leaks are simply detected or also prevented.

Stevens et al. in [24] present a comprehensive study on thirteen popular ad providers for Android. In particular, part of this study focuses on the analysis of ad traffic in order to detect the transmission of the user’s PII. The authors observe that at the time of writing only one of the considered ad providers leverage encryption to protect its network traffic. For this reason, they choose to perform a deep packet inspection to identify the leakage of the user’s private information. The results show that several types of PII (e.g., age, gender, GPS position) are leaked in clear by ad libraries. Moreover, the authors highlight that although none of the considered ad providers is able to build a complete profile of the user, the presence of UDIDs in ad-related traffic can be exploited by an external adversary to correlate sensitive information from different ad providers and build a complete user profile.

Kuzuno and Tonami in [24] investigate the leakage of sensitive information by the advertisement libraries embedded into free Android apps. They focus on both original and hashed identifiers unique to mobile devices (i.e., IMEI and Android ID) and SIM cards (i.e., IMSI and SIM Serial ID), as well as on the name of the cellular operator (CARRIER). The authors develop two components: (i) a server application; and (ii) a mobile app that can be installed on an Android device. The server application takes in input the network traffic of a set of apps that leak sensitive information, and applies a clustering method (see Section VI-C3 for details) to generate traffic signatures. The mobile app leverages such signatures to identify the sensitive information leaked by the other apps installed on the device. To evaluate their solution, the authors employ the network traffic of 1188 free Android apps and achieve the following results: 94% of HTTP messages containing sensitive information are correctly detected, with 5% false negatives (i.e., undetected HTTP messages carrying sensitive information), and less than 3% false positives (i.e., HTTP messages without sensitive information, incorrectly identified as sensitive). Since the signature generation phase requires to inspect HTTP messages looking for sensitive information, the system cannot work on encrypted traffic.

Rao et al. in [26] and Ren et al. in [49] present ReCon, a cross-platform system that allows mobile users to control the PII leaked in the network traffic of their devices. ReCon is based on Meddle (we described it in Section III-B), therefore it can inspect mobile traffic even if it is encrypted at transport layer, but cannot cope with data transmissions protected by IPsec. Moreover, ReCon offers a web interface through which the user can visualize in real time which PII is leaked, and optionally modify such PII or block the connection carrying it. In [26], the authors target two mobile OSes, namely Android and iOS, and the PII leakage detection mechanism is based on a domain blacklist. In [49], also Windows Phone is considered, and PII leaks are detected using properly trained machine learning classifiers, among which the best-performing (98.1% accuracy) is a C4.5 Decision Tree (more details in Section VI-C3). The works in [26], [49] expose an extensive leakage of sensitive information belonging to all the types of PII we listed above, as well as the transmission of usernames and passwords in both plain-text (HTTP) and encrypted (HTTPS) traffic.

Le et al. in [36] present AntMonitor, a system for collecting and analyzing network traffic from Android devices (we already mentioned it in Section III-B). Among other types of analysis, AntMonitor can perform PII leakage detection. The authors capture the network traffic of nine Android users for a period of five weeks, then inspect the collected dataset searching the following PII: IMEI, Android Device ID, phone number, email address, and device location. Overall, 44% and 66% of the analyzed apps leak IMEI and Android Device ID, respectively, while PII related to the user is rarely disclosed to remote hosts. It is worth to notice that the proposed analysis requires to inspect application-layer data, which is infeasible in case of traffic encryption, either at network (IPsec) or transport layer (SSL/TLS).

Song and Hengartner in [39] develop PrivacyGuard, an open-source Android app that leverages the VPNService class of the Android API for eavesdropping the network traffic of the apps installed on the device. The authors employ PrivacyGuard to investigate the leakage of PII related to mobile users (e.g., phone number) and devices (e.g., IMEI) by Android apps. In an evaluation conducted using 53 Android apps, PrivacyGuard detect more PII leaks than TaintDroid [65]. The proposed app can optionally replace such PII or block the connection carrying it. Moreover, it can inspect transmission protected by SSL/TLS (through a man-in-the-middle approach), but cannot deal with traffic encrypted via IPsec.

Vanrykel et al. in [52] investigate the leakage of sensitive identifiers in the unencrypted network traffic of Android apps. The authors develop a framework that automatically executes apps, collects their network traffic, inspects the HTTP data, and detects the identifiers that are transmitted in clear. The analysis of 1260 Android apps (from 42 app categories) shows that: (i) the Android ID and Google Advertising ID are the most frequently leaked identifiers, while the SIM serial

| Year | Paper | Android | iOS | Windows Phone | Symbian |
|------|-------|---------|-----|---------------|---------|
| 2014 | Chen et al. [28] | ✓ | ✓ | | |
| 2015 | Coull et al. [29] | ✓ | ✓ | | |
| 2016 | Ruffing et al. [49] | ✓ | ✓ | ✓ | ✓ |
| 2017 | Malik et al. [52] | ✓ | ✓ | ✓ | |
number, the IMSI, the device serial number, and the email of the registered Google account are less common in apps’ network traffic; (ii) there is an extensive leakage of app-specific identifiers; and (iii) certain apps leak the user’s phone number, email address, or position.

Continella et al. in [55] develop Agrigento, an open-source framework for the analysis of Android apps in order to detect PII leakage. Agrigento is based on differential analysis, and its workflow consists of two phases. In the first phase, the app under scrutiny is executed several times on a physical device to collect: (i) its network traffic; and (ii) additional system- and app-level information that is contextual to the execution (e.g., randomly-generated identifiers, timestamps). Subsequently, the collected information is aggregated to model the network behavior of the app. In the second phase, a specific PII within the operating system of the mobile device is set to a different value. The app is then executed once again to collect its network traffic and the contextual information. Finally, a PII leakage is reported if the collected data does not conform to the model learned before. Evaluated on 1,004 Android apps, Agrigento detects more privacy leaks than currently available state-of-the-art solutions (e.g., ReCon [48]), while limiting the number of false positives. The proposed framework requires to inspect HTTP messages and leverages a man-in-the-middle approach to deal with HTTPS traffic. However, Agrigento does not work on network traffic encrypted via IPsec.

### G. Sociological Inference

A property of a mobile device (e.g., the list of installed apps, the Wi-Fi networks to which the device associated) characterizes its owner. Sociologists can leverage this kind of information to study a population of mobile users. In this section, we review the works that deal with sociological inference, which we define as the analysis of the network traffic generated by mobile devices in order to infer some kind of sociological information about their users.

Barbera et al. in [23] investigate whether sociological information about a large crowd can be inferred by inspecting the Wi-Fi probe requests generated by the mobile devices of those people. First of all, the authors: (i) devise a methodology to convert a dataset of Wi-Fi probe requests into a social graph representing the owners of the monitored mobile devices; and (ii) develop an automatic procedure to infer the language of a given SSID. Subsequently, they target gatherings of people at urban, national, and international scale, as well as a mall, a train station, and a campus. In what follows, we summarize the authors’ findings: (i) the social graph of all the targeted events has social-network properties; (ii) the distributions of languages and mobile device vendors match the nature of the monitored crowds; and (iii) socially interconnected people tend to adopt mobile devices of the same vendor, and appear in the same time slot.

#### H. Tethering Detection

The ability to connect to cellular networks lets mobile devices have nearly ubiquitous Internet connectivity. Moreover, mobile devices are able to share such connectivity with other devices that cannot leverage cellular networks (e.g., laptops). This practice is commonly referred to as tethering, and can be carried out in many ways, such as via a USB cable, via Bluetooth, or establishing a WLAN (hotspot) for which the mobile device acts as a router.

In this section, we report the works that deal with tethering detection, which we define as the analysis of the network traffic generated by a mobile device in order to discover if it is sharing its Internet connection with other devices. This type of analysis can be valuable for a cellular network provider, since tethering can significantly increase the amount of traffic its network infrastructure has to sustain. An effective detection method would let the cellular ISP prevent users from sharing their mobile Internet connection, or require them to pay an extra fee.

Chen et al. in [28] develop a probabilistic classifier being able to detect tethering by leveraging several network features (e.g., the number of distinct TTLs in the packets coming from the same IP address). To evaluate their solution, the authors use publicly available Wi-Fi traces collected at two conferences, as well as a dataset of Wi-Fi traffic from a campus network. To simulate tethering, packets from different IP addresses are randomly mixed, then their source IP addresses are properly modified. On the public traces, the classifier reaches 0.68-0.85 recall with target precision fixed at 0.95, and 0.78-0.89 recall with target precision fixed at 0.8, outperforming alternative methods based on decision trees and linear regression. On the campus traces, the classifier reaches 0.86 precision, 0.74 recall, and 0.8 F-measure. Since the proposed solution requires to inspect the content of TCP headers, it cannot work whether the captured network traffic is protected by IPsec.

### TABLE V

| Year | Paper                 | Targeted Mobile Platform | Action on PII Leaks |
|------|-----------------------|--------------------------|---------------------|
| 2012 | Stevens et al. [20]   | ✓                        | ✓                   |
| 2013 | Kuzuno et al. [21]    | ✓ ✓                      | ✓ ✓                 |
|      | Rao et al. [20]       | ✓ ✓ ✓                    | ✓                   |
| 2015 | Le et al. [36]        | ✓ ✓ ✓ ✓                  | ✓ ✓ ✓               |
|      | Song et al. [39]      | ✓ ✓ ✓ ✓                  | ✓                   |
| 2016 | Ren et al. [38]       | ✓ ✓ ✓ ✓                  | ✓ ✓ ✓               |
|      | Vanuyke et al. [37]   | ✓ ✓ ✓ ✓ ✓                | ✓ ✓ ✓               |
| 2017 | Continella et al. [55]| ✓ ✓ ✓ ✓ ✓                | ✓ ✓ ✓               |
Network management can benefit from knowing the properties of the Internet traffic that traverse the network. Such information can be used to efficiently deploy the hardware equipments, as well as to setup them in order to provide the best Quality of Service (QoS) to the users. This statement particularly holds for networks serving mobile devices, since such devices generate traffic with peculiar properties. In light of the rapid evolution of mobile devices, the characterization of their Internet traffic is crucial to provide network administrators the information they need for resource planning, deployment, and management.

We define as traffic characterization the analysis of the network traffic generated by mobile devices in order to infer its properties. We group the works that deal with traffic characterization into two sub-categories, according to the scope of the analysis:

- The works that study the network traffic of specific apps and/or mobile services. We survey nine works belonging to this category. Rao et al. in [14] study the Android and iOS native apps of two video streaming services, namely Netflix and YouTube. YouTube is targeted also in [11]. The Android apps of Facebook and Skype are considered in [15]. In [4], Wei et al. focus on 27 Android apps (19 free and 8 paid). In [37], the analysis covers over 1,000,000 unique Android apps. In [34], Chen et al. analyze 5560 malicious Android apps (from 177 malware families). In [47], Nayam et al. study 63 Android and 35 iOS free apps, all belonging to the “Health & Fitness” category. The work in [5] focuses on five interactive apps for both Android and iOS. In [50], Espada et al. present a framework for traffic characterization of Android apps, and choose Spotify as case study.

- The works that study the network traffic generated by a population of mobile devices. We can further divide such works into two subsets:
  - The works that compare mobile traffic with non-mobile one. We survey four works belonging to this subcategory. The work in [9] focuses on the network traffic of mobile devices when they are connected to home Wi-Fi networks, while the works in [6], [12], [13] carry out the same analysis for campus Wi-Fi networks.
  - The works that only consider mobile traffic. We survey four works belonging to this subcategory. The works in [16], [59] target campus Wi-Fi networks, while the works in [7], [10], [35] leverage client-side measurements collected through logging apps.

In Table VI, we provide a view of the classification described above. In what follows, we list the main properties of mobile traffic that stem from the works we survey:

- At the network layer, IP flows of mobile devices have shorter duration, much higher number of packets, and much smaller packets, compared to IP flows of non-mobile devices [13].
- Most of the transport-layer traffic is carried over TCP [7], [12], [16], and more than half is encrypted [7]. Transfers within TCP connections are small in size [7], [12], causing a high overhead for lower-layer protocols, particularly when transport-layer encryption is in place [7].
- Most of the application-layer traffic is carried over HTTP or HTTPS [7], [12], [13], [36], [31], [34], [59]. Moreover, the analysis carried out by Chen et al. in [16] shows that: (i) the adoption of HTTPS is increasing (a trend confirmed in [47], [59]); and (ii) Akamai and Google servers serve nearly 40% of the mobile traffic.
- Mobile devices contact a less diverse set of hosts compared to non-mobile devices [12], [16].
- Mobile devices experience a loss rate on Wi-Fi networks [16]. On cellular networks, instead, there are high delays and losses, as well as low throughputs [7].
- An important fraction of the mobile traffic is due to video streaming [9], [12], mainly on the YouTube platform [11].
- Android apps typically do not encrypt their network traffic (simply relying on HTTP), connect to several different hosts, and a fraction of their network traffic is related to Google’s services [4].
- A significant part of the network traffic generated by Android and iOS free apps is due to advertisement and tracking services [47].
- The Android apps of Netflix and YouTube tend to periodically buffer large portions of the video to be played, while their iOS counterparts tend to initially buffer a large amount of data, then periodically buffer small portion of the video to keep playback ongoing (although the YouTube app employs large-block buffering under favorable network conditions) [14]. Moreover, probably to deal with the TCP timeouts caused by the delays of cellular networks, the Netflix and Youtube iOS apps create a large number of TCP flows to provide a single video, thus causing an overhead that is not necessary when mobile devices are connected to Wi-Fi networks [14], [16].

J. Trajectory Estimation

In Section II-C, we discussed the applications and privacy implications of device positioning. In this section, we present a different problem: to infer the trajectory followed by a mobile device in a geographical area, by analyzing the network traffic it generates. This type of traffic analysis, which we call trajectory estimation, can help study the interests and social habits of a mobile user (if we focus on a single individual), as well as aid traffic prediction along urban streets (if we aggregate the trajectories of several mobile users).

Musa and Eriksson in [18] present a system for passively tracking mobile devices by leveraging the Wi-Fi probe requests they periodically transmit. The idea is to employ a number of Wi-Fi monitors, which look for probe requests from mobile devices and report each detection to a central server, where the detections of the same mobile device are turned into a spatio-temporal trajectory. To evaluate their system, the authors set up
three deployments and leverage GPS ground truth to measure the accuracy of the inferred trajectories. The mean error is under 70 meters when the distance among the monitors is over 400 meters.

K. Usage Study

The habits of mobile users have significantly changed with the evolution of cellphones toward smartphones and tablets. First of all, the adoption of the touchscreen display has completely revolutionized the human-machine interaction. Moreover, the development of mobile operating systems supporting multitasking and third-party apps has enhanced the capabilities of mobile devices well beyond the requirements for communication activities. In this scenario, it is fundamental to understand how mobile users interact with their mobile devices in order to improve the usability of mobile OSes and apps, as well as to properly set up networks serving mobile devices. For instance, the knowledge of places where mobile devices are mostly used can drive the deployment of free Wi-Fi hotspots in order to reduce the traffic load on cellular networks.

We define as usage study the analysis of the network traffic of mobile devices in order to infer the usage habits of mobile users. The works we review in this section leverage network-side measurements [6], [9], [11], [12], [13], [17], [25], as well as data collected within mobile devices [7], [17], [25], [28]. Overall, we identify three perspectives to study the usage habits of mobile users:

- The network. As an example, we can study when users are active (i.e., sending and receiving data) during the day, how long their periods of activity are, how much traffic they generate, and which are the most frequently used network interfaces (i.e., Wi-Fi or cellular).
- The apps and/or mobile services. As an example, we can study which are the most frequently used apps/services, and which is the traffic volume of a specific app.
- The geographical positions and mobility patterns. As an example, we can study where mobile devices are most frequently used, and where they generate most of their traffic.

In Table VI, we show the types of usage study carried out in the surveyed works that deal with this kind of traffic analysis. In what follows, we summarize their findings:

- The most frequently used apps are the ones related to multimedia content (e.g., YouTube, Spotify) and web browsers [7], [9], [12], [13], [17], [25]. Social network and instant messaging apps are also popular [25].
- The predominance of cellular over Wi-Fi network traffic observed for mobile devices by Ham and Choi in 2012 [17] is gradually disappearing. As reported in [25], in 2015 more than half of mobile traffic is carried over Wi-Fi. In particular, mobile users tend to switch to Wi-Fi connectivity whenever a Wi-Fi access point is available [25].
- The usage of mobile devices is low at nighttime and high in daytime [6], [13], [17], [28]. Cellular traffic peaks in commute times, while Wi-Fi traffic peaks in the evening [25]. Cellular traffic is lighter on weekends than weekdays, while Wi-Fi traffic follows the opposite trend [25].
- Mobile users tend to generate more network traffic when they are out of home, and when their devices have high battery level [28].
- The volume of traffic generated by the mobile users of a Wi-Fi network varies greatly, from less than 100 MB to several GBs, according to users’ habits and needs [25].
- According to Finamore et al. in [11], YouTube users on mobile devices: (i) similarly to non-mobile users, they prefer short videos (40% of the watched videos are shorter than three minutes, and only 5% are longer than ten minutes); (ii) similarly to non-mobile users, they rarely change video resolution and, whenever they do that, it is to switch to a higher resolution (although full screen mode is not frequently used); and (iii) more frequently than non-mobile users, they early stop watching the video (within the first fifth of its duration for 60% of the videos).
TABLE VII
THE SURVEYED WORKS THAT DEAL WITH USAGE STUDY.

| Year | Paper | Network | Apps/Mobile Services | Geography/Mobility |
|------|-------|---------|----------------------|--------------------|
| 2010 | Afanasyev et al. [18] | ✔ | ✔ | ✔ |
|      | Falaki et al. [11] | ✔ | ✔ | ✔ |
|      | Maier et al. [9] | ✔ | ✔ | ✔ |
| 2011 | Finamore et al. [11] | ✔ | ✔ | ✔ |
|      | Gember et al. [12] | ✔ | ✔ | ✔ |
|      | Lee et al. [13] | ✔ | ✔ | ✔ |
| 2012 | Ham et al. [17] | ✔ | ✔ | ✔ |
| 2015 | Fukuda et al. [35] | ✔ | ✔ | ✔ |
|      | Soikkeli et al. [15] | ✔ | ✔ | ✔ |
| 2017 | Wei et al. [59] | ✔ | ✔ | ✔ |

L. User Action Identification

As we stated in Section III-B, most of the apps can leverage the Wi-Fi and cellular network interfaces of mobile devices to send and receive data. Since users perform several actions while interacting with apps, it is likely that most of such actions generate data transmissions. The network traffic trace of a given action typically follows a pattern that depends on the nature of the user-app interaction of that action. As a practical example, browsing a user’s profile on Facebook will likely produce a different traffic pattern compared to posting a message on Twitter. These patterns can be used to recognize specific user actions related to a particular app of interest in generic network traces. Moreover, it is often possible to infer specific information about a given user action (e.g., the length of the message sent via an instant messaging app). We define as user action identification these types of traffic analysis.

The possibility to identify actions of mobile users can be useful in several scenarios:
- It is possible to profile the habits of a mobile user (e.g., checking emails in the morning, watching YouTube videos in the evening). The user’s behavioral profile can be used to later recognize the presence of that user in a network. Moreover, profiles of thousands of mobile users can be aggregated in order to infer some information for marketing or intelligence purposes.
- It is possible to perform user de-anonymization. Suppose a national agency is trying to discover the identity of a dissident spreading anti-government propaganda on a social network. It is possible to monitor a suspect and detect when she posts messages via the social network mobile app. The inferred posting timestamps can be matched with the time of the messages on the dissident social profile in order to understand if the suspect is actually the dissident.

In Table VIII, we show the app categories covered by the surveyed works that deal with user action identification [27], [29], [37], [41], [50]. Almost all the works target communication apps, which belong to the most privacy-sensitive app category. This category includes instant messaging apps (e.g., iMessage, KakaoTalk, WhatsApp) as well as email clients (e.g., Gmail, Yahoo Mail). Another sensitive category is social (e.g., Facebook, Twitter, Tumblr), which is targeted in [44], [50]. Apps related to multimedia contents (e.g., YouTube) are considered in [27], [50]. Moreover, Saltaformaggio et al. in [50] also focus on other categories of apps: dating (e.g., Tinder), health (e.g., HIV Atlas), maps (e.g., Yelp), news (e.g., CNN News), and shopping (e.g., Amazon). The works in [27], [44] cover productivity apps (e.g., Dropbox), and Watkins et al. in [27] also consider mobile games (e.g., Temple Run 2) and utility apps (e.g., ZArchiver).

Watkins et al. in [27] develop a framework that exploits the inter-packet time of responses to ICMP packets (i.e., pings) to infer the type of action that the target user is performing on her mobile device. In particular, the authors focus on three types of user action: (i) CPU intensive; (ii) I/O intensive; and (iii) non-CPU intensive. First of all, the authors check the feasibility of their approach for the Android and iOS platforms, showing that unfortunately their solution does not work for the latter because iOS does not use CPU throttling. Subsequently, they evaluate their framework using six Android apps, achieving a minimum 93% accuracy. Since the proposed solution exploits the timing of packets, it is not affected by traffic encryption.

Coull and Dyer in [29] target iMessage, Apple’s instant messaging service, which is available as an app for iOS or a computer application for OS X. The proposed analysis leverages the sizes of the packets exchanged between the target user and Apple’s servers, thus it works despite all iMessage communications are encrypted. The authors focus on five user actions: “start typing”, “stop typing”, “send text”, “send attachment”, and “read receipt”. Assuming to have correctly inferred that the target mobile device is running iOS, all user actions can be classified with over 99% accuracy, except for the “read receipt” action that is often confused with the “start typing” action. The authors also aim to infer the language (among six languages: Chinese, English, French, German, Russian, and Spanish) and length of the exchanged messages. Assuming to have correctly identified an iMessage action on a mobile device running iOS, the language classification achieves more than 80% accuracy by considering the first 50 packets. Besides, the length classification achieves an average error of 6.27 characters for text messages, and an absolute error of less than 10 bytes for attachment transfers.

Park and Kim in [37] target KakaoTalk, an instant messaging service widely used in Korea. They consider eleven actions that a user can perform on the Android app (e.g., join a chat room, send a message, add a friend). For each action, the proposed framework learns its traffic pattern as a sequence of packets. Such sequence is then used to recognize that specific action in unseen network traces. The proposed solution reaches 99.7% accuracy despite KakaoTalk traffic is encrypted.
Conti et al. in [44] present an identification framework which leverages the information available in IP and TCP headers (e.g., source and destination IP addresses) and therefore it works even if the network traffic is encrypted via SSL/TLS. However, the proposed approach does not work on an IPsec scenario, since it relies on (IP address, TCP port) pairs to separate traffic flows. The authors target seven popular sites and domains (i.e., Google+, Tumblr, and Twitter), reaching over 95% accuracy and precision for most of the actions, and outperforming websites fingerprinting algorithms by Liberatore and Levine (2016).

Saltaformaggio et al. in [50] present NetScope, a user action identification system that can be deployed at Wi-Fi access points or other network equipments. Since it leverages IP headers/metadata, NetScope can be employed even if the network traffic is protected by IPsec. The authors evaluate their solution by considering 35 user actions from 22 apps across two platforms (i.e., Android and iOS) and eight app categories. The identification accuracy reaches average precision and recall of 78.04% and 76.04% respectively, performing better for Android devices rather than iOS ones.

### M. User Fingerprinting

Mobile users interact actively with their devices, leveraging the nearly ubiquitous Internet connectivity and the capabilities of the apps available in the marketplaces. To each mobile user, it is possible to associate a set of preferred (i.e., most frequently used) apps and, for each of these apps, a set of preferred (i.e., most frequently executed) actions. Since most of the mobile apps are able to connect to the Internet, and many user actions within them trigger data transmission through the network, it becomes clear that the network traffic generated by a user is likely to present a fairly constant pattern across different devices, as well as across different networks. We define as user fingerprinting the attempt to exploit such pattern in order to recognize the network traffic belonging to a specific mobile user. This type of analysis can be applied to:

- Recognize the presence of a specific mobile user within a network. Once the network is identified, it is then possible to approximate the geographical position of that user with the location of the Wi-Fi hotspot or cellular station to which her mobile device is connected.
- Partition the network traces of a mobile traffic dataset by user. Once the transmissions related to a specific mobile user have been separated and grouped together, it is then possible to apply other types of traffic analysis targeting that user.

In this section, we review the works that deal with mobile user fingerprinting.

Vanrykel et al. in [52] investigate how mobile unencrypted traffic can be exploited for user surveillance. The authors develop a framework to automatically execute apps, collect their network traffic, inspect the HTTP data, and identify the sensitive identifiers that are transmitted in clear. Moreover, the authors present a graph building technique that exploits such identifiers to extract the network traces generated by a specific mobile user from a traffic dataset (more details in Section VI-B1). The analysis of 1260 Android apps (from 42 app categories) shows that the proposed solution can link 57% of a mobile user’s unencrypted network traffic. In addition, the authors observe the limited effectiveness of ad-blocking apps in preventing the leakage of sensitive identifiers.

Verde et al. in [53] present a system being able to accurately infer when a target user is connected to a given network and her IP address, even though she is hidden behind NAT among thousands of other users. To achieve this objective, a machine learning classifier is trained with the NetFlow records of the target user’s traffic, and then employed to analyze the NetFlow records of a given network in order to detect the presence of the target user within it (more details on the classifier are provided in Section VI-C10). The system is evaluated as follows:

- Cross-validation is applied to the NetFlow records of the traffic generated by 26 different mobile users connecting to the Internet through the same Wi-Fi access point. The best performing implementation of the classifier (which is based on random forest) reaches 95% true-positive rate, 7% false-positive rate, 0.95 precision, 0.93 recall, and 0.94 F-measure.

### TABLE VIII

| Year | Paper | Communication | Dating | Gaming | Health | Maps | Media | News | Productivity | Shopping | Social | Utility |
|------|-------|---------------|--------|--------|--------|------|-------|------|--------------|----------|--------|---------|
| 2013 | Watkins et al. [27] | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| 2014 | Conti et al. [44] | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| 2015 | Park et al. [37] | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| 2016 | Saltaformaggio et al. [50] | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
It is worth to notice that the proposed solution is encryption-agnostic, since NetFlow records can be extracted even from encrypted traffic.

### N. Website Fingerprinting

The Internet has a central role in people’s everyday life, and surfing the Web has become a common task that can be performed from desktop computers, as well as in mobility using laptops and smartphones/tablets, thanks to the increasing deployment of cellular and Wi-Fi networks. From a privacy point of view, the set of websites frequently visited by a user is a sensitive information, since it can disclose her interests, social habits, religious belief, sexual preference, and political orientation.

In the field of traffic analysis, website fingerprinting generally indicates the attempt to infer the website or even webpage visited by a user surfing the Internet, by analyzing the network traffic generated by her web browser. This type of analysis has been extensively treated in the domain of personal computers, where machine learning techniques have been proved to be very effective \cite{66}, \cite{67}, \cite{68}. Since we focus on mobile devices, in this section we survey the works that target users navigating through the web browser of their mobile devices.

Spreitzer et al. in \cite{51} develop an Android app being able to capture the data-usage statistics of the browser app, and leverage them to fingerprint the webpages visited by the user of the mobile device on which that app is installed. This solution is not affected by encryption, since it only requires to know the amount of data transmitted and received by the browser app (which is easily obtainable in Android). The proposed app is evaluated on a set of 500 possible pages that the user can visit. Overall, 97\% of 2,500 page visits are correctly inferred. The authors also evaluate their fingerprinting app when the network traffic is protected by Tor (in particular, by using the Orbot proxy and the Orweb browser). Out of a set of 100 possible pages that the user can visit, 95\% of 500 page visits are correctly inferred.

### IV. Points of Capturing for Traffic Analysis Targeting Mobile Devices

Another meaningful categorization for the works in the field of traffic analysis is according to the point where the network traces are captured in order to build the traffic dataset. In Table IX we report the point(s) of capturing for each surveyed work. As shown in Figure 4 the most common points of capturing are wired network equipments (21 works) and mobile devices themselves (twenty works), followed by Wi-Fi access points (eleven works), Wi-Fi monitors (five works), and machines running mobile device emulators (four works). In one work, the mobile traffic is simulated via software. As shown in Table IX four works leverage two different types of point of capturing, and one work even three. In the following sections, we present the point(s) of capturing for each surveyed work. We also discuss the effect of encryption (e.g., IPsec, SSL/TLS) on the collected datasets and performed analyses.

| Year | Paper | Mobile Device Emulators | Mobile Devices | Simulator | Wired Network Equipments | Wi-Fi Access Points | Wi-Fi Monitors |
|------|-------|------------------------|---------------|-----------|------------------------|--------------------|--------------|
| 2010 | Afanasyev et al. \cite{6} | ✓ | ✓ | ✓ | ✓ |
| 2011 | Falaki et al. \cite{21} | ✓ | ✓ | ✓ | ✓ |
| 2012 | Rusten et al. \cite{12} | ✓ | ✓ | ✓ | ✓ |
| 2013 | Lee et al. \cite{13} | ✓ | ✓ | ✓ | ✓ |
| 2014 | Rao et al. \cite{14} | ✓ | ✓ | ✓ | ✓ |
| 2015 | Spreitzer et al. \cite{51} | ✓ | ✓ | ✓ | ✓ |
| 2016 | Ren et al. \cite{48} | ✓ | ✓ | ✓ | ✓ |
| 2017 | Arora et al. \cite{54} | ✓ | ✓ | ✓ | ✓ |
A. Mobile Devices

The most direct way to collect mobile traffic is to place the point of capturing within the mobile devices, leveraging their modern operating systems to run a full-fledged logging app that is able to gather the required information. This approach has several implications:

- The covered set of mobile devices tends to be small compared to network-side measurements, since the logging app has to be installed on the mobile device of each volunteer.
- Since the traffic is logged directly on the mobile device, we are sure that everything is captured belongs to that mobile device. This is an advantage because in case of network-side logging, the mobile traffic needs to be separated from the transmissions generated by other kinds of device, such as laptops and desktop computers. This process is error-prone, since some network information could be potentially misclassified. Moreover, it lacks completeness, because traffic that is not classified for some reason will be discarded even if it belongs to a mobile device.
- The logging app must have the proper permissions to capture traffic on the mobile device.
- The logging app must be lightweight. This means that it has to: (i) impose a negligible computational burden on the mobile device’s CPU; (ii) occupy as few memory as possible to store traffic logs (this problem is easy resolvable if the logging app is allowed to upload the logs to a remote server); and (iii) cause minor battery consumption, which is a major concern for mobile users.
- It is possible to focus on the traffic generated by specific mobile apps, or the one transiting through a specific network interface (i.e., Wi-Fi or cellular).

In Table X, we present the works in which the mobile traffic is captured within one or more mobile devices. For each work, we show the targeted mobile platforms, the number of mobile devices employed, the tool used to capture the network traffic, and the information leveraged for the analysis.

Android is the most targeted mobile platform, mainly because its open nature makes it easy to develop a traffic logger from scratch [69], or simply port one of the available desktop solutions. Nevertheless, Fukuda et al. in [35] and Shepard et al. in [10] show that effective traffic loggers can be successfully deployed also on iOS devices. Shark for Root is based on tcpdump, while tPacketCapturePro leverages the VPNService class of the Android libraries, which is also used in the custom logging apps by Le et al. in [50] and Song and Hengartner in [39].

The number of targeted devices is a meaningful information only for works which carry out mobile traffic characterization [2], [10], [4], [33], [56], or study the usage habits of mobile users [7], [17], [35]. With regard to such works, we observe that only Fukuda et al. in [35] leverage a suitable population (over 1500 mobile devices), while the others count not more than a few tens of mobile devices. We point out that the problem is less relevant for the works in [4], [56], since the focus of the analysis is on the apps, rather than mobile devices.

Regarding network traffic encryption, we provide the following observations:

- Network traffic statistics (e.g., the amount of received bytes through the cellular network) are not affected by encryption, therefore the methods that leverage them (namely the ones proposed by Ham and Choi in [17], Shabtai et al. in [19], Shabtai et al. in [32], Fukuda et al. in [35], Spreitzer et al. in [51], and Arora and Peddouju in [54]) are encryption-agnostic. It is worth to notice that this assertion is not trivial for the work in [51], since its authors leverage the TCP bytes sent/received by the browser app, which should be theoretically unavailable if the traffic is encrypted via IPsec. However, mobile browsers are very unlikely to employ IPsec for communicating with web servers.
- Falaki et al. in [7] carry out both traffic characterization (Section III-I) and usage study (Section III-K). The former is focused on the TCP protocol, therefore it does not cover the traffic protected by IPsec, whenever present in the collected network traces. The latter leverages the per-app transmitted/received bytes, which are network traffic statistics (i.e., they are not affected by encryption).
- Shepard et al. in [10] provide a few findings about the network traffic of iOS devices. Their analysis is focused on the TCP protocol, so it does not consider the network traffic protected by IPsec.
- Su et al. in [21] propose a classifier for Android malware detection that cannot process the network traffic encrypted via IPsec, since one of the leveraged features is the average TCP session duration, which is not computable without accessing TCP headers.

- To characterize the network traffic of the considered Android apps, Wei et al. in [4]: (i) inspect the IP addresses of the captured packets; (ii) compute the amount of transmitted/received data; and (iii) discriminate between HTTP and HTTPS traffic. The first two actions are encryption-agnostic, while the third one is not possible in case an app employs IPsec to hide its network transmissions. However, it seems that the analyzed apps do not use such type of encryption.
- To apply clustering for PII leakage detection, Kuzuno and Tonami in [24] use two metrics that are based on information within HTTP messages. Such information is not accessible if the network traffic is encrypted.
To evaluate their solution for Android app identification, Qazi et al. in [25] set up a monitored access point serving a few mobile devices. The network traffic flowing through the AP is captured, and netstat logs from the devices are gathered. Such logs are then used to match the network flows observed at the AP with the TCP transmissions from the mobile devices. This methodology does not work if the apps on the mobile devices employ IPsec to communicate.

- The user action and OS identification methods devised by Coull and Dyer in [29] are designed to work with the mobile traffic of iMessage (Apple’s instant messaging service), which is encrypted by default.

- Le et al. in [36] carry out both app identification (see Section III-B) and PII leakage detection (see Section III-F). The former requires to access the flags of TCP segments, which are hidden if IPsec is employed. The latter needs to inspect application-layer data, which is inaccessible if the traffic is encrypted.

- The app for PII leakage detection by Song and Hengartner [39] employs a man-in-the-middle approach to inspect TLS traffic, but it cannot deal with IP packets whose payloads are encrypted by IPsec.

- The Android malware detection solution by Zaman et al. [42] needs to access the URLs within HTTP messages, which are not available if the traffic is encrypted.

- Mongkollucksamee et al. in [45] extract the TCP and UDP data from the collected network traffic of the apps to be profiled. After that, they inspect the headers to reconstruct the captured network flows and compute their statistics (e.g., per-flow total amount of transferred bytes). This approach cannot be applied to apps that encrypt their network traffic using IPsec.

- The malware detection framework by Narudin et al. [46] requires to inspect HTTP messages, therefore it cannot work with encrypted traffic.

- By employing the mobile traffic characterization framework proposed by Espada et al. [56], it is possible to check whether the network traffic of an Android app satisfies a given property. The effect of encryption on the analysis depends on the properties to be verified. Regarding the presented case study (Spotify), the authors successfully access HTTP headers and compute traffic statistics (e.g., number of sent/received TCP segments).

B. Wired Network Equipments

In this section, we review the works in which the mobile traffic is captured at one or more wired network equipments. Such equipments can be deployed into two types of network:

- Small-scale networks, serving a reduced number of mobile users (from one single mobile device to a few tens). Researchers often deploy such networks to collect the traffic they need in a controlled environment. The equipments associated to this type of network are small Internet gateways [33], VPN servers [26], [48], [52], and traditional desktop computers that log all the traffic traversing the wired link between the APs to which the mobile devices are associated and the Internet [14], [15], [22], [34], [37], [44], [47], [53], [55], [58]. The user population typically consists only of the targeted mobile devices (i.e., there is no need to filter out non-mobile traffic from the captured traces).
• Large-scale networks, serving thousands of users. In such case, the considered network equipments are edge routers (i.e., routers that connect customers to their ISP’s backbone) [9], top-level routers [13], [18], [33]. Internet gateways [6], [28], [59], switches [28], or generic points of presence within national ISPs and campus networks [11].

The user population typically includes also non-mobile users (e.g., laptop users), and the network traffic they generate must be removed from the captured traces. The works in which the mobile traffic is extracted from traces captured at one or more wired network equipments serving small-scale and large-scale networks are presented in sections [IV-B1] and [IV-B2] respectively.

1) Small-scale Networks: In Table [XI] we present the works in which the collected mobile traffic comes from one or more wired network equipments serving a small number of mobile devices. For each work, we report the network equipments at which the mobile traffic is logged, the targets of the analysis, additional details about the capturing process, and the information leveraged for the analysis. We use the term forwarding server to indicate a device that logs all the traffic traversing the wired link between the APs to which the monitored mobile devices are connected and the Internet.

In the remaining of this section, we clarify how the works reported in Table [XI] deal with encryption. Rao et al. in [14] study the network traffic of the Android and iOS apps for Netflix and YouTube. They successfully inspect the HTTP messages to get the encoding rate of the videos, therefore both services stream videos in clear (at least, they did so at the time the authors collected their dataset). The analysis carried out by Baghel et al. in [15] needs to inspect the transport-layer headers, therefore it does not work if IPsec is employed to hide the payload of IP packets. The Android malware detector by Wei et al. [22] requires to access DNS data, which is not possible if the traffic is encrypted. To carry out PII leakage detection, Rao et al. in [26] and Ren et al. in [48] inspect HTTP traffic, which is sent in clear, and also HTTPS traffic, which is decrypted using SSL split. This approach cannot work, however, if the traffic of a given app is protected by IPsec. The mobile user fingerprinting framework presented by Verde et al. in [33] takes NetFlow records as input. Since NetFlow records can be extracted even from encrypted traffic, the proposed solution is encryption-agnostic. Chen et al. in [34] focus on the properties of the network traffic of malicious Android apps, and their findings are mainly related to the application layer. For this reason, such findings are limited to the data that the analyzed apps sent in clear during the capturing process. The user action identification frameworks developed by Conti et al. in [44], and Park and Kim in [37], and the app identification solution proposed by Taylor et al. in [58], leverage the information available in IP and TCP headers. As a consequence, such approaches are by design resilient against SSL/TLS, but cannot cope with encryption via IPsec. To study the network behavior of several Android and iOS free apps, Nayam et al. in [47] inspect the HTTP messages, and employ a proxy server to deal with HTTPS traffic. Although such approach does not work with apps that employ IPsec to hide their network transmissions, it seems that all the analyzed apps do not leverage such type of encryption. The PII leakage detection and user fingerprinting framework proposed by Vanrykel et al. in [52] is focused on unencrypted mobile traffic only, since it requires to inspect HTTP messages. Wang et al. in [53] present two Android malware detection models which leverage TCP- and HTTP-related information, respectively. The latter cannot work for apps that encrypt their network traffic using SSL/TLS, and both cannot cope with apps that employ IPsec for their data transmissions. The PII leakage detection solution by Continella et al. [55] requires to access the HTTP messages. Although a man-in-the-middle approach is adopted to deal with HTTPS traffic, the framework cannot cope with network traffic protected by IPsec.

2) Large-scale Networks: In Table [XII] we present the works in which the mobile traffic is extracted from traces captured at one or more wired network equipments serving large-scale networks with thousands of users. In such case, for each work we provide additional information about the mobile data extraction process and the effects of encryption in the considered scenario.

a) Mobile Data Extraction: Afanasyev et al. in [6] leverage the Organizationally Unique Identifier (OUI) of the MAC address to discriminate between mobile and non-mobile devices. This approach has the disadvantage that an OUI can be associated to devices of both types (e.g., several OUIs belonging to Apple are associated to iPhones and MacBooks as well). Maier et al. in [9] and Chen et al. in [16] inspect the User-Agent field of HTTP messages, which can be misleading and it is not present in non-HTTP mobile traffic. In such case, the authors in [9] inspect the Time-To-Live (TTL) field of IP packets. Finamore et al. in [11] leverage the peculiar characteristics of YouTube traffic from mobile devices, while Lee et al. in [13] inspect the packet headers, looking for information related to mobile operating systems (without clarifying the nature of such information). Chen et al. in [28] do not elicit mobile traffic from the gathered network traces because they simply merge such data with traffic from other sources (see Section [IV-D] for details), then simulate tethering by modifying the source IP address of packets. Verde et al. in [33] do not need to extract the traffic of mobile devices from the collected network data because identifying such traffic is just the goal of their user fingerprinting method. Wei et al. in [59] inspect the IP address to check whether it belongs to the IP address pool of the target WLAN, then leverage the DHCP logs from the DHCP server of the network to map the IP address to a MAC address, and finally inspect the Organizationally Unique Identifier (OUI) of the MAC address.

b) Encryption: Afanasyev et al. in [6] focus part of their analysis on the applications that generate mobile and non-mobile traffic. In particular, they need to inspect transport- and application-layer headers. For this reason, the reported findings do not cover the encrypted traffic present in the collected network traces. The same holds for the mobile traffic characterization by Chen et al. [16]. The analysis carried out by Maier et al. in [9] requires to access transport- and application-layer information, therefore it cannot deal with encrypted traffic. The study by Finamore et al. in [11] focuses on YouTube traffic carried over HTTP and does not consider the
users that watch videos via a secure connection (i.e., HTTPS). The app identification via payload signatures proposed by Lee et al. in [13] cannot work with apps that encrypt their network traffic. Moreover, in case of encryption the authors’ studies of mobile traffic characteristics and mobile users’ habits are severely limited. The tethering detection technique proposed by Chen et al. in [28] requires to inspect the information available in TCP headers, therefore it does not work if IPsec is employed to hide the payload of IP packets. As we previously mentioned in Section IV-B1, the mobile user fingerprinting framework by Verde et al. in [33] is encryption-agnostic even in their experiment carried out on a large-scale network, since it takes NetFlow records as input. A few of the findings about mobile traffic reported by Wei et al. in [39] are based on application-layer information, which is unavailable in case of traffic encryption.

C. Wi-Fi Access Points

As reported in [31], mobile users are increasingly offloading their traffic demands to Wi-Fi networks. This practice has become very common at home, where Wi-Fi modems are employed to make the wired Internet connection of the house available to laptops and mobile devices. Moreover, free Wi-Fi networks are often deployed in shops and public places (e.g., parks, malls, train stations), as well as at social events (e.g., meetings, conferences, concerts).

A Wi-Fi network typically consists of two types of hardware equipments: (i) access points (APs), which leverage the 802.11 standard to provide network connectivity to the associated

| Year | Paper | Network Equipment(s) | Target(s) | Capturing Details | Leveraged Information |
|------|-------|-----------------------|-----------|-------------------|----------------------|
| 2011 | Rao et al. [14] | Forwarding server | Android and iOS clients for Netflix and YouTube | 180 seconds for each video playback | HTTP messages |
| 2012 | Baghel et al. [15] | Forwarding server | Facebook Android app | 90 minutes, no user interaction | Layer-2+ data |
|      |       |                       | Skype Android app | Five hours, no user interaction | DNS data |
|      | Wei et al. [22] | Forwarding server | 102 malicious Android apps | No details | DNS data |
| 2013 | Rao et al. [26] | VPN server | The top 100 free Android apps in Google Play Store, and 209 iOS apps from Apple App Store | Up to ten minutes of manual interaction | Layer-3+ data |
|      |       |                       | 732 free Android apps from a third-party Android marketplace | Android Debug Bridge (ADB) scripting and Monkey are leveraged to execute 100,000 actions for each app | |
| 2014 | Verde et al. [33] | Gateway router | 26 mobile devices | One month of monitoring | NetFlow records |
| 2015 | Chen et al. [34] | Forwarding server | 5560 malicious Android apps | Each app is stimulated for five minutes using Monkeyrunner | Layer-2+ data |
|      | Park et al. [37] | Forwarding server | Eleven user actions from KakaoTalk Android app | Each user action is automatically executed 100 times | IP headers, TCP headers |
|      |       |                       | 58 user actions from seven Android apps | Android Debug Bridge (ADB) scripting is leveraged to execute 220 sequences of actions for each app | IP headers, TCP headers |
| 2016 | Conti et al. [44] | Forwarding server | 63 Android and 35 iOS free apps, all belonging to the “Health & Fitness” category | The execution of the apps is driven using automated scripts (using Appium for Android and Silk Mobile for iOS) | HTTP messages |
|      | Nayam et al. [37] | Forwarding server | The top 100 free apps for Android, iiOS, and Windows Phone | Five minutes of manual interaction | Layer-3+ data |
|      | Ren et al. [48] | VPN server | 8/50 of the top 1,000 free apps from a third-party Android marketplace | Android Debug Bridge (ADB) scripting and Monkey are leveraged to execute 10,000 actions for each app | |
|      | Vanykel et al. [52] | Two VPN servers | 1200 Android apps (from 42 app categories) | User interactions are simulated using The Monkey | HTTP messages |
|      | Wang et al. [53] | Forwarding server | 8,312 Android benign apps, and 5,560 Android malware samples | Each app is stimulated using Monkeyrunner | TCP- and HTTP-related data |
| 2017 | Continella et al. [55] | Forwarding server | 1,004 Android apps | Each app is stimulated for ten minutes using Monkey | HTTP messages |
|      | Taylor et al. [58] | Forwarding server | 110 Android apps | Android Debug Bridge (ADB) scripting is leveraged to simulate user-app interactions | IP headers, TCP headers |
| Year | Paper                  | Targeted Network Equipment(s) | Number of Monitored Users | Capturing Period | Leveraged Information                                                                                           | Methodology Applied to Extract Mobile Data                                                                 |
|------|------------------------|-------------------------------|---------------------------|-----------------|----------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------|
| 2010 | Afanasyev et al. [6]   | Central Internet gateway of an urban Wi-Fi network | Over 2500 simultaneous    | 5 days          | Layer-3+ headers (excluding DHCP data[^7]) of the first packet of each flow for the first quarter of each hour | Leverage RADIUS logs from the APs of the network to map each IP address observed at the gateway to a MAC address, then inspect the Organizationally Unique Identifier (OUI) of the MAC address |
|      | Maier et al. [9]       | Edge router of an ISP's network | Over 20,000               | 4 days over 11 months | Anonymized DSL data                                                                                              | Inspect the User-Agent field of HTTP messages or, for non-HTTP traffic, inspect the Time-To-Live (TTL) field of IP packets |
| 2011 | Finamore et al. [11]   | Five points of presence within national ISPs and campus networks | Unspecified               | 1 week          | IP packets                                                                                                      | Inspect the URL requests, looking for app=youtube_gdata or app=youtube_mobile                                |
|      | Lee et al. [13]        | Top-level router of a campus network | Unspecified               | 6 days          | IP packets                                                                                                      | Inspect packet headers, looking for information related to mobile operating systems                           |
| 2012 | Chen et al. [16]       | Gateway router of a campus network | Unspecified               | 3 days, 1 day   | Up to 900 bytes of each incoming/outgoing packet, including IP, TCP, and application-level headers | Inspect the IP address (since separate IP pools are used for Ethernet and WLAN), then inspect the User-Agent field of HTTP messages |
| 2014 | Chen et al. [28]       | Wired switch serving the APs of a Wi-Fi network | Unspecified               | Unspecified    | Layer-2+ headers, plus DHCP and DNS payloads (all data are anonymized)                                         | None                                                                                                          |
|      |                        | Internet gateways of a campus Wi-Fi network | 12,600                    | 1 week          | IP packets                                                                                                      | None                                                                                                          |
|      | Verde et al. [33]      | Tier-2 router of a metropolitan Wi-Fi network | 200,000                   | 1 day           | NetFlow records                                                                                                 | None                                                                                                          |
| 2017 | Wei et al. [59]        | Internet gateway of a campus Wi-Fi network | 6,482                     | 1 month         | Layer-3+ data                                                                                                   | Inspect the IP address to check whether it belongs to the WLAN IP address pool, leverage the DHCP logs from the DHCP server to map the IP address to a MAC address, and inspect the Organizationally Unique Identifier (OUI) of the MAC address |

[^7]: The DHCP requests are handled by the APs of the network, thus they do not reach the central Internet gateway.
wireless devices; and (ii) gateways, which forward the network traffic coming from the APs to the Internet (or to an higher-level gateway, in case of a hierarchical network infrastructure), and vice versa. It is worth to notice that these two categories are not mutually exclusive, since a hardware equipment can act as both access point and gateway (e.g., Wi-Fi modems). In this section, we survey the works that apply analysis methods to mobile traffic captured at one or more Wi-Fi access points (we dealt with traffic capturing at gateways in Section IV-B).

From an analysis point of view, we make the following observations:

- Compared with cellular networks, Wi-Fi networks cover a smaller geographical area, as well as much less users. For this reason, the mobile traffic captured at the APs of a Wi-Fi network is representative of a more restricted user population (e.g., the customers of a shop, the students of a campus), enabling fine-grained analysis.
- Since Wi-Fi networks are typically free of charge, mobile users can carry out intensive network activities (e.g., watching videos from a streaming platform). Such kind of activities are hard to observe in cellular networks due to the fees applied to the Internet traffic by the network providers.
- Wi-Fi networks usually serve not only mobile devices, but also other kinds of device, such as desktop computers and laptops. Therefore, if the analysis targets mobile devices, the network traffic belonging to non-mobile devices must be properly filtered out from the collected network traces.
- If the monitored Wi-Fi network employs several access points, the information gathered at each AP must be properly combined with the one from the other APs, in order to produce a comprehensive network trace. This process can be tricky whenever the APs (or the traffic sniffers deployed at the APs) are not perfectly synchronized, causing timestamps from different sources to be staggered.

In Table XIV, we present the surveyed works in which the mobile traffic is captured at one or more Wi-Fi access points. We identify two distinct sets: (i) the works in which the authors monitor a single access point, deployed in a controlled environment to provide Internet connectivity to a small number of mobile devices [20, 25, 27, 28, 41, 43, 50, 5, 57] (more details in Section IV-C1); and (ii) the works in which several APs of a real Wi-Fi network serving a large number of users are monitored [6, 12] (more details in Section IV-C2).

1) Single AP in Controlled Environment: Stevens et al. in [20] study thirteen popular ad libraries for Android. For each library, the authors build a simple app that makes ad requests, then they execute it on a mobile device while capturing the network traffic at the access point to which that device is associated. Since only one of the considered ad libraries leverages encryption to protect its network traffic, the authors apply deep packet inspection to investigate the leakage of the user’s PII. Qazi et al. in [25] set up a wireless access point running OpenFlow and instruct it to extract features from the network traffic of the associated mobile devices. Since it requires to inspect transport-layer information, the proposed framework cannot process network traffic protected by IPsec. The framework for user action identification presented by Watkins et al. in [27] exploits the inter-packet time of the responses to ICMP packets sent to the target mobile device by a laptop connected via cable to the same network, therefore it is not affected by traffic encryption. The OS identification method by Chen et al. [28] needs to access the headers of TCP segments, so it cannot work if IPsec is employed to hide the payload of IP packets. The app identification solution by Yao et al. [41] requires to access HTTP messages, which is not possible if the traffic is encrypted. The app identification framework proposed by Alan and Kaur [43] provides three different classifiers. Two of them only leverage the size of IP packets, thus they can take encrypted traffic as input. Instead, the other classifier requires to inspect the content of TCP headers, therefore it works on network traffic encrypted via SSL/TLS, but it does not via IPsec. The solutions proposed by Saltaformaggio et al. in [50] (user action identification), as well as Tadrous and Sabharwal in [5] (traffic characterization), are encryption-agnostic: the former requires only to inspect IP headers; the latter needs only the size and header information of 802.11 frames. The framework for OS identification presented by Malik et al. in [57] exploits the inter-packet time of the packets (either ICMP responses or IP packets related to video streaming) coming from the target mobile device, therefore it is not affected by traffic encryption.

2) Multiple APs in Real Deployment: Afanasyev et al. in [6] leverage encryption-agnostic data-link- and network-layer information from the RADIUS logs collected by the over 500 APs of the Google Wi-Fi network in Mountain View (California, USA), and discriminate among desktop computers, laptops, and mobile devices by relying on the Organizationally Unique Identifier (OUI) of the MAC address. Gember et al. in [12] carry out a comparison between mobile and non-mobile devices with regard to network traffic properties and habits of users. To discriminate between the two types of device, they apply the following methodology: (i) for a device generating HTTP traffic, the User-Agent field of the HTTP messages is compared with a list of strings clearly related to mobile devices, in order to determine if the device is a mobile one (match) or not (mismatch); (ii) for a device that does not generate HTTP traffic, the Organizationally Unique Identifier (OUI) of its MAC address is inspected to determine whether it is a mobile or non-mobile device. Since the analysis is mainly focused on transport and application layers, most of the authors’ findings are related to non-encrypted traffic.

D. Wi-Fi Monitors

We define a Wi-Fi monitor as a hardware equipment that is able to scan the Wi-Fi radio bands (i.e., 2.4 and 5 GHz) in order to capture the transiting IEEE 802.11 frames. The most common configuration consists of a traditional Wi-Fi device (e.g., a PCI card in a desktop computer) set in monitor mode, i.e., the device passively listens the nearby Wi-Fi transmissions. To effectively eavesdrop the network traffic of a Wi-Fi device, the monitor must be within the target’s range of transmission. Such range depends on many factors, including the selected radio band, the power of the Wi-Fi module, and the surrounding buildings.
Wi-Fi monitors can be easily deployed at a low cost, and let oversee a good number of Wi-Fi devices. However, there are a few issues that have to be addressed in order to effectively use Wi-Fi monitors for eavesdropping:

- In case more than one monitor is deployed, an IEEE 802.11 frame can be eavesdropped by multiple distinct monitors if they are too close to each other. When traffic traces provided by different monitors are merged, the duplicate captures must be properly deleted.

- The timestamp of each eavesdropped IEEE 802.11 frame depends on the internal clock of the Wi-Fi monitor that captured it. Since the network data collected by distinct monitors are merged to build a comprehensive dataset, it is crucial to consider internal clock differences between monitors (unless they are synchronized in some way).

In Table XIV we provide information about the capturing process carried out in the works that employ one or more Wi-Fi monitors to collect the network traffic of mobile devices. In what follows, we discuss two additional issues, and how the works reported in Table XIV address them:

- The traffic from non-mobile devices must be filtered out from the captured network traces. In [18] and [23], the capturing takes place in a location and time such that the collected traffic only belongs to mobile devices. In [40] and [49], the network data generated by non-mobile devices is filtered out since the MAC address of each targeted mobile device is known.

- 802.11 frames content can be protected by encryption. This is not a concern for Musa and Eriksson in [18] and Barbera et al. in [23], since their frameworks focus on probe requests, which are sent in clear. The analyses carried out by Wang et al. in [40] and Ruffing et al. in [49] are encryption-agnostic, since they leverage only size and/or timing of the captured 802.11 frames. However, this statement does not hold for Chen et al. in [28], since their analysis requires to access IP payloads.

### E. Mobile Device Emulators

As for their desktop counterparts, apps must be properly tested not only throughout their development process, but also before their final submission to a marketplace. The simplest testing methodology consists of installing the mobile app on one or more physical mobile devices. However, this approach has two shortcomings: (i) the number of different test configurations (each consisting of a hardware device and a version of a compatible mobile operating system) is limited, while the number of configurations available on the market is large; and (ii) the difficulty in automating the tests within the mobile devices, due to lack of tools and resource constraints.

An alternative solution is to install the app to be tested in a mobile device emulator, which is a virtual machine that is able to simulate the components and operations of a mobile operating system. This approach has several implications:

- The Software Development Kit (SDK) of a mobile platform typically provides a mobile device emulator for testing purposes. Therefore, the developers can cut the expense for the physical mobile devices and simply buy a machine to run the emulator (or, even better, run the emulator directly on the machine where the code is written, without any additional expenditure).

- If properly endowed with computation power and memory, the testing machine can run multiple mobile device emulators in parallel, speeding up the overall testing process or letting the developers increment the set of tests to be executed on the app.

- A mobile device emulator can be quite easily controlled from the outside, helping test automation and thus reducing human intervention.

- There are important limitations on which components and operations of a mobile operating system can be emulated. Such limitations reduce the types of test that can be actually executed on a given app.

Since a machine running a mobile device emulator is responsible for forwarding the network traffic from the emulator to the Internet and vice versa, it constitutes an ideal point of capturing for mobile traffic. This approach is particularly useful if the focus of the analysis is on the network traffic of a specific mobile app. In Table XV we present the targeted mobile platforms and number of considered apps for the surveyed works in which the mobile traffic is captured at one or more machines running mobile device emulators.

Crussell et al. in [30] carry out ad fraud detection on two sets of Android apps: (i) 130,339 apps crawled from nineteen different marketplaces; and (ii) 35,087 apps that probably are malware (provided by an unspecified security company). The

### TABLE XIII

| Scale of the Targeted Wi-Fi Network(s) | Year | Paper | Leveraged Information |
|-------------------------------------|------|-------|-----------------------|
| Single AP in controlled environment | 2012 | Stevens et al. [20] | HTTP messages |
|                                     | 2013 | Qi et al. [25] | Network- and transport-layer information |
|                                     | 2014 | Watkins et al. [27] | Inter-packet time of ICMP responses |
|                                     | 2015 | Yao et al. [31] | IP headers, TCP headers |
|                                     | 2016 | Chen et al. [28] | Layer 2+ information |
|                                     | 2017 | Malik et al. [57] | Inter-packet time of ICMP responses or IP packets related to video streaming |
| Multiple APs in real deployment    | 2010 | Afanasyev et al. [6] | Data-link- and network-layer information from RADIUS logs |
|                                     | 2011 | Gember et al. [12] | Layer 2+ information |
TABLE XIV
THE SURVEYED WORKS IN WHICH THE MOBILE TRAFFIC IS CAPTURED USING ONE OR MORE Wi-Fi MONITORS.

| Year | Paper            | Number of Wi-Fi Monitors | Capturing Duration | Targeted Population                                      | Leveraged Information               |
|------|------------------|--------------------------|--------------------|-----------------------------------------------------------|-------------------------------------|
| 2012 | Musa et al. [18] | 5                        | 9 months           | People along the streets near an university campus        | 802.11 probe requests               |
|      |                  | 6                        | 12 hours           | People along fairly busy roads of a city                  |                                     |
|      |                  | 7                        | 12 hours           | People along an arterial road of a city                   |                                     |
| 2013 | Barbera et al. [23] | 5                        | From 40 minutes to 7 hours | People at two political meetings, two Pope's masses, a big mall, and a train station | 802.11 probe requests               |
|      |                  | 1                        | 6 weeks            | People at an university campus                            |                                     |
|      |                  | 1                        | Unspecified        | People at streets and aggregation places of a city        |                                     |
| 2014 | Chen et al. [28] | 9                        | 2 days             | People at OSDI 2006                                      | Size and header of IP packets       |
|      |                  | 8                        | 3 days             | People at SIGCOMM 2014                                    |                                     |
| 2015 | Wang et al. [40] | 1                        | Unspecified        | Unspecified number of iOS devices                         | Size and timing of (possibly encrypted) 802.11 frames |
| 2016 | Ruffing et al. [49] | 1                        | 3 months           | Two Android devices, two iOS devices, a Windows Phone device, and a Symbian device | Timing of (possibly encrypted) 802.11 frames |

TABLE XV
THE SURVEYED WORKS IN WHICH THE MOBILE TRAFFIC IS CAPTURED AT ONE OR MORE MACHINES RUNNING MOBILE DEVICE EMULATORS.

| Year | Paper            | Number of Apps Run in Mobile Device Emulator(s) |
|------|------------------|-----------------------------------------------|
|      |                  | Android | iOS | Symbian |
| 2014 | Crussell et al. [30] | 165,420 | None | None |
|      | Lindorfer et al. [31] | Over 1,000,000 | None | None |
| 2015 | Yao et al. [41]  | 651,000 | 68,000 | 10,000 |
| 2016 | Narudin et al. [46] | 1030 | None | None |

authors apply the following modus operandi for capturing the network traffic of a given app: (i) the app is installed on a newly created Android emulator image; (ii) a logger starts to capture the emulator’s network traffic; and (iii) the app is first run in the foreground for 60 seconds, then it runs in the background for another 60 seconds. The proposed framework is not resilient to encryption, since it needs to inspect the HTTP and DNS data generated by apps. However, the authors’ analysis covers most of the available ad libraries. This means that such libraries do not usually employ any form of encryption for their data transfers, and simply rely on plain HTTP.

Lindorfer et al. in [31] present ANDRUBIS, a publicly available system for the analysis of Android apps. For each submitted app, ANDRUBIS applies both static and dynamic analysis techniques in order to study how the app behaves. Moreover, during the 240 seconds of the dynamic analysis, the network traffic generated by the app running in the sandbox is captured for a later analysis focused on high-level protocols (e.g., DNS, HTTP, IRC). Unfortunately, such analysis is not feasible if the app encrypts its network traffic.

Yao et al. in [41] carry out app identification on three mobile platforms, namely Android, iOS, and Symbian. To capture network traffic from the selected apps, the authors run them in mobile device emulators and trigger their network behavior using UI automation tools (the framework also supports the capturing at a Wi-Fi access point, see Section [IV-C]). Since the system requires to inspect HTTP messages, it does not work if an app leverages HTTPS or lower-layers encryption.

Narudin et al. in [46] leverage machine learning to build a classifier being able to detect malware on the Android platform. They consider two sets of Android apps: (i) the top twenty free (benign) apps available in the Google Play Store; and (ii) 1,000 malicious apps from 49 malware families, provided by the Android Malware Genome Project, as well as 30 new (in 2013) malicious apps from fourteen malware families, collected by the authors. To capture the traffic of the malicious apps, two online dynamic analysis platforms, namely Anubis and SandDroid, are leveraged (the traffic of the benign apps is logged on a real device, see Section [IV-A]). The proposed analysis requires to inspect HTTP messages, hence it cannot take encrypted network traffic as input.

F. Simulator

In this section, we deal with works in which the mobile traffic is not captured from real mobile devices or mobile device emulators, but instead it is generated by a software simulator. This approach can be useful to study particular deployments of mobile devices that are not observable in a real-world scenario due to technical difficulties, economical constraints, or limits imposed by law. If the simulation is realistic, the resulting network traces will be really close to the ones that are collected on real/emulated mobile devices.

Network traffic simulation typically works as follows:
1) The information about the simulated environment (e.g., geographical extension, buildings, streets) is provided to the system.

2) The information about the actors (e.g., mobile devices, laptops, access points) is provided to the system. For each actor, such information includes its technical specifications, its position within the simulated environment, and its network behavior. If the actor is used by a human user, her sociological characteristics and behavioral patterns are also provided.

3) The points of capturing are positioned within the simulated environment.

4) The network transmissions of the actors are simulated according to realistic physical laws and social dynamics.

Husted and Myers in [8] develop a 3D simulation of a large population of mobile devices deployed in a dense metropolis where no other Wi-Fi devices (e.g., access points) are present. A fraction of the mobile devices act as trackers (i.e., they are the points of capturing) and scan the air in order to capture Wi-Fi probe requests transmitted by the rest of the population (i.e., the trackees). The system properly simulates the propagation of probe requests (which are transmitted in clear) in the environment, and takes into account the diurnal behavior of mobile users (e.g., go to work in the morning, come home in the evening). The resulting network traffic dataset is leveraged for trajectory estimation (more details in Section III-J).

V. Targeted Mobile Platforms in Traffic Analysis

The network traffic of a mobile device depends on its operating system. Since each mobile OS has its own implementation of the network protocol stack, it generates data transmissions with peculiar network properties. Exploiting such properties is fundamental to devise effective methods for the analysis of mobile traffic. For example, the TCP window size scale option (i.e., the value that is negotiated during the TCP three-way handshake to increase the TCP receiver window size beyond 535 bytes) is always 16 for iOS, while it can be either 2, 4, or 64 for Android. Chen et al. in [28] exploit this distinction (together with other differences with regard to network traffic) to successfully recognize whether a target mobile device is running one of those OSes.

In this section, we present the surveyed works according to the mobile platforms they target. As shown in Figure 5, only thirteen works propose analyses that are platform-independent, i.e., they do not take into account the platform which the targeted mobile devices belong to (it is worth to notice that two of them, namely Lee et al. in [13] and Chen et al. in [28], also present other types of analysis that are instead tailored to specific mobile platforms). Among the other works, Android is the most targeted mobile platform (42 works), followed by iOS (fifteen works), Windows Mobile/Phone (four works), and Symbian (two works). As shown in Table XVI, nine works target two mobile platforms, three works target three platforms, and one work even four. Each of the following sections is dedicated to a mobile platform: Android (Section V-A), iOS (Section V-B), Symbian (Section V-C), and Windows Mobile/Phone (Section V-D). Each of the above-mentioned sections is organized in three parts: the first part provides an overview of the system architecture; the second part describes the apps specific for that platform; and the third part reviews the works that carry out traffic analysis targeting the such platform. Finally, we report works that do not belong to any specific mobile platform in Section V-E.

### A. Android

Android is an open-source mobile operating system developed by Google. Android is also promoted by the Open Handset Alliance (OHA), a consortium of 84 firms (including Google, as well as several important actors of the mobile
market, like HTC, Samsung, and LG) which is devoted to the development of open standards for mobile devices. Android was unveiled at the end of 2007, and the first batch of commercial Android devices appeared a year later. Many mobile device manufacturers soon started deploying Android on their flagship products, and the popularity of the operating system rapidly increased. Nowadays, Android is the dominant mobile operating system, with a market share of 68.4% in June 2016, according to the statistics reported in [70].

1) **System Architecture:** As shown in Figure 6a, the architecture of the Android operating system consists of a stack of four abstraction layers:

1) At the first layer, a Linux kernel provides system services (e.g., memory, power and process management), preemptive multitasking, a network stack and hardware devices drivers (e.g., display, camera).

2) The second layer contains the Android Runtime (ART), which is the application runtime environment. Before Android 5.0 (Lollipop), the execution of Android apps is managed by the process virtual machine Dalvik. Android apps and services are typically written in Java and executed in a Dalvik Virtual Machine after being converted from Java Virtual Machine to Dalvik bytecode. ART adopts a different approach: the Dalvik bytecode is translated into native instructions to be later executed on the runtime environment of the device. This solution increases efficiency and reduces power consumption. This layer also includes native libraries that provide several functionalities (e.g., 2D/3D graphics, encryption, SQLite database management).

3) The third layer is the application framework, the environment that runs and manages Android apps. Among the available services that compose such environment, (i) the Activity Manager manages app lifecycle and activity stack; (ii) the Content Providers allow apps to share data with other apps; (iii) the Telephony Manager interfaces with telephony services available on the device; (iv) the Notifications Manager prompts the user with notification or alerts raised by apps; and (v) the Location Manager provides the apps with periodic updates regarding the location of the device.

4) The forth layer is constituted by the apps, which can be native (e.g., web browser, email client) or provided by a third party.

2) **Apps:** Android apps run in a sandbox and their access to each system’s resource is regulated by a specific permission that has to be given by the user. Before Android 6.0 (Marshmallow), an app’s required permissions are presented to the user at the beginning of installation process. The user must grant all the required permissions in order to install the app on her device. From Android 6.0 on, permissions are managed individually, and users can grant or revoke each permission according to their usability and security needs.

A third-party Android app is shipped in an APK (Android application package) file, which can be downloaded from the developer’s website and manually installed on the device. To simplify the process, Android users typically rely on the app stores, or app marketplaces, which are programs that allow them to browse the available apps, as well as to install, update, and remove them. Google Play Store (formerly Android Market) is the primary app store installed on Android devices, and hosts over 2,500,000 apps [71] distributed by both Google itself and third-party developers under Google’s license and compatibility requirements. However, the openness of Android has allowed the birth of a number of other third-party app marketplaces (e.g., GetJar, F-Droid, the app store by Amazon), which release apps under policies different from Google’s one.

3) **Traffic Analysis:** Since mobile apps are a key component of the success of the Android operating system, it is not surprising that most of the surveyed works focus their analysis on the network traffic generated by Android apps. The achieved results show that it is possible to successfully fingerprint an Android app (or type of app) [13], [25], [26], [36], [41], [43], [45], [58], as well as an action performed by a mobile user on her Android device [27], [37], [44], [50]. In [26], [36], [39], [43], [52], [55], it is reported that Android apps extensively leak the PII of mobile users, and the works in [20], [24] highlight that an important role in this phenomenon is played by the embedded ad libraries. Regarding mobile advertisement, Crussell et al. in [30] prove that many Android apps trick the advertisement business model in order to let their developers illicitly earn money. In [52], the authors exploit the sensitive identifiers that are present in mobile traffic to fingerprint Android users. In [51], the network statistics of Android’s default web browser are leveraged for website fingerprinting. Finally, several works aim at detecting malicious Android apps; in [22], [42], [46], [53], [54], automated detection frameworks that can be employed by marketplaces and security companies are presented; instead, the authors in [19], [32] present apps that can enable malware detection directly within the mobile devices of the end users.

In light of its market share, we argue that Android is the reference operating system for many mobile users, and Android devices are responsible for an important fraction of the worldwide mobile Internet traffic. For this reason, it is not surprising that many works aim at studying the properties of the network traffic generated by Android devices [7], [14], [15], [4], [31], [34], [35], [47], [5], [56], as well as the usage habits of Android users [7], [17], [35]. Moreover, Android

![Fig. 5. Number of published works contributing traffic analysis methods targeting mobile devices, sorted by mobile platform.](image-url)
plays an important role in the works that deal with mobile OS identification [28], [49], [57].

B. iOS

iOS is a proprietary mobile operating system developed by Apple. Such OS is exclusively deployed in Apple’s mobile devices. iOS was officially released with the name iPhone OS in 2007. Later, this mobile OS was extended to support other Apple’s mobile devices: iPod Touch (Apple’s multimedia player) in 2007 and iPad (Apple’s tablet) in 2010. According to the statistics reported in [70], iOS is the second most popular mobile operating system, with a market share of 20.32% in June 2016.

1) System Architecture: As shown in Figure 6b, the architecture of the iOS operating system consists of a stack of four abstraction layers, each providing different services and technologies:

- The Core OS layer contains: (i) the kernel; (ii) the device drivers; (iii) the interfaces to access the low-level features of the operating system (e.g., file-system, memory, concurrency, networking); and (iv) the interfaces to access the frameworks that provide several core functionalities (e.g., support for external hardware, Bluetooth, authentication, cryptography, support for VPN tunnels).

- The Core Services layer includes the mandatory system services for running apps. These services provide core functionalities (e.g., account management, location services, cellular network services), as well as high-level features (e.g., P2P, data protection, file sharing, SQLite, XML).

- The Media layer contains technologies leveraged by developers to implement multimedia content in their apps (i.e., audio, video and graphic).

- The Cocoa Touch layer provides the key frameworks which define the appearance of apps and grant the access high-level system services (e.g., push notifications, touch-based input, multi-tasking).

2) Apps: Apple distributes the iOS Software Development Kit (SDK), which contains the tools needed to develop, test, and deploy native iOS apps. Apps are written in Objective-C or Swift, and leverage the iOS system frameworks. Such framework provides the interfaces that developers need to write software for the iOS platform. Apps are physically installed on the devices, and run directly on their operating system.

Third-party iOS apps are available to users in the App Store, Apple’s digital distribution platform, which was launched in 2008. The apps are developed with the iOS SDK and released after Apple’s approval. The review process aims at assessing that the distributed apps fulfill precise usability and security requirements. According to the statistics reported in [72], the App Store hosts about two million apps, available for various iOS devices (e.g., iPhone, iPad). It is worth to notice that there exist also unofficial marketplaces that distribute iOS apps (e.g., Cydia), but they all require a jailbroken iOS device. In a jailbroken iOS device, software vulnerabilities have been exploited to remove the restrictions imposed by iOS. This practice is required to allow the download and installation of apps, extensions, and themes that are unavailable through the official Apple App Store.

3) Traffic Analysis: Mobile apps are a fundamental building block of the iOS user experience. For this reason, several solutions have been proposed to effectively fingerprint them [13], [26], [40], [41], as well as to detect the interactions between an iOS user and a specific app installed on her mobile device [29], [50]. The authors in [26], [48] investigate the disclosure of sensitive information by iOS apps, discovering that many of them leak the PII of the user. Regarding OS identification, Coull et al. in [29] discriminates between iOS and OS X, while the frameworks presented in [28], [49], [57] consider iOS among the targeted mobile operating systems. Finally, a few works study the properties of the network traffic generated by iOS devices [10], [14], [45], [47], [5], and the usage habits of iOS users [35].

C. Symbian

Symbian is a mobile operating system originally developed for PDAs in 1998, and subsequently moved into cellphones and smartphones in the following years. Running exclusively on ARM processors, Symbian requires an additional middleware to form a complete operating system and to provide a user interface. During the 2000s, Symbian became the most popular mobile OS, since many mobile manufacturers, particularly Nokia, chose it to power their devices. A non-profit organization, the Symbian Foundation, was created in 2008 to drive the development of the operating system and promote the adoption of Nokia’s middleware, namely S60. However, with the advent of Android and iOS, and Nokia adopting Windows Phone for its devices, the popularity of the Symbian platform rapidly decreased. The Symbian Foundation
closed in 2010, and the development of the OS ended in that period. According to the statistics reported in [70], Symbian is almost disappeared, with a market share of only 2.22% in June 2016.

1) **System Architecture:** As shown in Figure [6c], the architecture of the Symbian operating system consists of a stack of three abstraction layers:

1) The OS layer is the core of a Symbian system, and contains the kernel, which provides the interfaces to access the underlying hardware, and several essential services (e.g., communications, text and data handling, graphics).

2) The Middleware layer provides a software platform which consists of higher-level generic APIs available to the apps of the upper layer. These APIs include the native UI frameworks, as well as frameworks for app lifecycle, higher-level protocols, and data handling. Different platforms are not compatible, i.e., apps developed for a platform cannot run on the others.

3) The Apps layer includes apps that interact with the user and background services that provide functionalities to the apps.

2) **Apps:** As we already explained in Section [V.C], all Symbian devices share a common core, on top of which different software platforms are built to provide an execution environment for user apps (actually implementing the Middleware layer shown in Figure [6c]). Backed by different groups of mobile device manufacturers, three software platforms were created for Symbian:

- S60 (Series 60) was the most popular Symbian platform, officially supported by the Symbian Foundation and deployed in the products of several mobile device manufacturers, including Nokia, Samsung, and LG. S60 was able to run apps developed in Java MIDP, C++, Python, and Adobe Flash. Third-party developers had to distribute their apps by either releasing them in the marketplaces (the most important stores were run by Nokia and Opera Software), or pre-installing them in the mobile devices of some manufacturers.

- UIQ (User Interface Quartz) was developed by UIQ Technology, and supported by Sony Ericsson and Motorola. The platform was able to run native apps written in C++ using the Symbian/UIQ Software Development Kit (SDK), as well as Java apps. The development of UIQ stopped in 2008, when the Symbian Foundation was established and chose S60 as its reference Symbian platform.

- MOAP (Mobile Oriented Applications Platform) was the platform chosen by NTT DoCoMo, a major Japanese cellular operator, for its FOMA (Freedom of Mobile Multimedia Access) service, which was a W-CDMA-based 3G telecommunications service. Supported by a few Japanese companies, like Fujitsu and Sharp, MOAP did not spread outside of Japan. It was not an open development platform, i.e., there were no third-party apps.

3) **Traffic Analysis:** Only a pair of works we survey do target the Symbian operating system. The first work by Ruffing et al. [49] deals with the identification of the OS of mobile devices, and Symbian is among the operating systems that the proposed framework is able to recognize. The second work by Yao et al. [41] presents an app identification system which is trained and evaluate on, among others, 10,000 Symbian apps from the Nokia OVI Store.

**D. Windows Mobile/Phone**

In the early 1990s, Microsoft began to develop a new operating system for minimalist computers and embedded systems. This OS, later called Windows CE and officially released in 1996, was the basis for the operating systems that make Microsoft enter into the mobile market at the beginning of 2000s. The first batch of mobile devices running a Microsoft’s OS were Windows Mobile smartphones. They became available in 2003 and targeted business users at first. The lifecycle of Windows Mobile lasted for approximately seven years, ending in 2010 with the release of its successor, Windows Phone, which had a new user interface and aimed at the consumer market. The last iteration of this OS was Windows Phone 8.1, released in 2014 and succeeded by Windows 10 Mobile at the end of 2015. Overall, Microsoft’s mobile OSes struggle to acquire a relevant market share and seem not to threaten the duopoly by Android and iOS (a trend confirmed by the fact that, according to the statistics reported in [70], only 1.94% of mobile devices were Windows Phone ones in June 2016).

1) **System Architectures:** As shown in Figure [6d], the architecture of the Windows Mobile operating system follows a stack model, consisting of three abstraction layers:

- The Original Equipment Manufacturer (OEM) Layer is positioned at the bottom of the stack. This layer directly communicates with the underlying hardware components (e.g., microprocessor, RAM, ROM, digital signal processors, input/output modules).

- The Operating System (OS) Layer includes the kernel, the core DLLs, the object store (which offers file system, registry, and database persistent storage), multimedia technologies, the device manager, communication and networking services, and the Graphic Windowing and Events Subsystem (GWES). The later one provides an interface between the OS, the app, and the user.

- The Application Layer consists of the apps, from either Microsoft itself or third parties.

As shown in Figure [6d], the architecture of the Windows Phone operating system is different, although it maintains the three-levels stack:

- At the bottom of the stack, the Software Foundation layer includes: (i) the kernel, which manages security, networking, and storage; and (ii) the interfaces that mediate the access to the underlying hardware components (e.g., sensors, camera).

- The intermediate layer is composed by three elements: (i) the App Model, which is the component providing first-class access to several functionalities that are important for apps (e.g., isolation, licensing, software updates, data sharing); (ii) the UI Model, which manages the user interface of the operating system; and (iii) the components that enable the integration with Microsoft’s cloud services.

- The Application Layer includes the frameworks available to developers for building the user interface and logic of their apps.
2) Apps: Apps for Windows Mobile are developed using the official Software Development Kit (SDK) released by Microsoft, and can be written either in C++ (“native” apps) or C#/Basic (“managed” apps). At the end of 2009, Microsoft set up a digital distribution platform, called Windows Marketplace for Mobile, to organize and centralize the release of apps for the Windows Mobile platform. With the advent of Windows Phone, Microsoft started to progressively abandon Windows Mobile, by ending support and closing Windows Marketplace for Mobile in 2012.

Although the SDK and the libraries are different, the apps for Windows Phone are written with the same languages used for Windows Mobile apps (i.e., C++, C#, and Basic), plus HTML5 and JavaScript for web-based apps. The official software distribution platform for Windows Phone, called Windows Phone Marketplace (and later renamed Windows Phone Store), was launched by Microsoft at the end of 2010, and subsequently merged into the Windows Store (i.e., Microsoft’s universal software marketplace) in 2015.

3) Traffic Analysis: Only a few works we survey target Microsoft’s mobile OSes. Falaki et al. in [2] deployed a custom logging app on Windows Mobile devices to capture their network traffic and study its properties. Ren et al. in [48] investigate PII leakages through network traffic generated by devices running several mobile operating systems, including Windows Phone. Finally, the works in [49], [57] deal with mobile OS identification, and Windows Phone is among the operating systems that the proposed frameworks are able to recognize.

E. Platform-independent Works

We survey several works in which the analysis performed on the network traffic is generic, which means that it is not specific for a particular mobile platform. Some of these works leverage the 802.11 probe requests that are sent by mobile devices of any platform to discover if an already known WiFi access point is nearby: in [23], sociological information is inferred from the probe requests of a population of mobile users, while in [16] and [18] probe requests are exploited to estimate a mobile device’s geographical position and movements, respectively. Instead, some work simply group together mobile devices of different platforms and consider them as a unique category. In [16], [59], the properties of the network traffic generated by mobile devices in a campus Wi-Fi network are studied. In [6], [9], [12], mobile and non-mobile devices are compared on network traffic properties and users’ usage habits, and the same is done in [11] but limited to the YouTube service. Finally, Verde et al. in [53] present a user fingerprinting method that is successfully used to recognize the presence of some target mobile users within a small test network and a large Wi-Fi one.

VI. Models and Methods in Traffic Analysis Targeting Mobile Devices

In this section, we provide a deeper insight into the models and methods leveraged in the state of the art to devise solutions for analysis of mobile devices’ network traffic. In Section VI-A, we describe a way to turn a dictionary into an effective classifier. In Section VI-B, we report a methodology to extract a social network from group of mobile users relying on their network traffic. Finally, in Section VI-C we deal with application of machine learning to several types of traffic analysis targeting mobile devices.

A. Dictionary

A dictionary can be used as a one-feature classifier whether: (i) the keys are the values that the feature can take; and (ii) a set of class labels is associated to each key. This solution is suitable to solve classification problems in which there is a single feature, which takes a limited set of values.

Coull and Dyer in [29] target iMessage, Apple’s instant messaging service, which is available as a mobile app for iOS or a traditional computer program for OS X. The objective is to fingerprint five distinct user actions (i.e., start typing, stop typing, send text, send attachment, and read receipt) by leveraging the size of the packets exchanged between the target iMessage client and Apple’s servers. The authors study the packet sizes corresponding to the considered user actions, and notice that each user action has two distinctive packet sizes: (i) one when a message is sent to Apple’s servers; and (ii) one when a message is received from Apple’s servers. This property can be exploited to build a classifier that takes the form of a dictionary in which one or more user action labels are associated to each packet size observed in the training data. When a new packet arrives, the dictionary is queried to retrieve the user action label(s) for its payload length: if only one label is found, the packet is given that label; if two or more labels are returned, the user action most frequently associated to that payload size during training is chosen. Once we correctly identify the OS on which iMessage is running (i.e., iOS or OS X), all the considered user actions are classified with accuracy exceeding 99%, except for the “read receipt” action that on iOS is easily confused with the “start typing” action.

B. Graph Analysis

In this section, we highlight a few works in which graph theory is leveraged for mobile traffic analysis. Barbera et al. in [23] combine graph theory and traffic analysis to carry out sociological inference targeting mobile users (their findings are reported in Section III-G). More specifically, they present a methodology to build the social network of a group of mobile users from the probe requests sent by their mobile devices. The proposed procedure consists of the following steps:

1) The dataset of collected probe requests is turned into an affiliation network. An affiliation network $G = (V_1, V_2, E)$ is a bipartite graph in which $V_1$ is a set of actors, $V_2$ is a set of groups the actors belong to, and each edge $e \in E$ connecting an actor $v_1 \in V_1$ to a group $v_2 \in V_2$ represents

---

1In this paper, we use the term dictionary to indicate a collection of (key, value) pairs, such that each key appears at most once in the collection, and a value can be either a single value or an unordered set of values.

2In a social network, the nodes correspond to the individuals, while the edges model the relationships between them.
a group membership. In the work by Barbera et al. [23], $V_1$ is the set of mobile devices (identified by their MAC address) that sent at least one probe request, $V_2$ is the set of SSIDs contained in the collected probe requests, and an edge $e \in E$ connecting a mobile device $v_1 \in V_1$ to an SSID $v_2 \in V_2$ represents $v_1$ having $v_2$ in its Preferred Network List (PNL), which is the list of the SSIDs of the Wi-Fi networks $v_1$ connected to in the past.

2) A similarity measure $f : V_1 \times V_1 \rightarrow R$ is chosen to represent the strength of the social relationship between the users of each pair of mobile devices $u$ and $v$. Based on the Adamic-Adar similarity measure [73], Barbera et al. [23] define the $f$ function as:

$$f(u, v) = \sum_{w \in N(u) \cap N(v)} \frac{1}{\log_2(|M(w)|)} \quad \text{(VI-B0.1)}$$

where $N(u)$ is the PNL of the mobile device $u$, and $M(w)$ is the set of mobile devices that have the SSID $w$ in their PNL.

3) The affiliation network $G$ is turned into a social network $G' = (V_1, E')$ by applying the following rule:

$$\forall u, v \in V_1 : (u, v) \in E' \iff f(u, v) > t \quad \text{(VI-B0.2)}$$

where $t$ is a minimum similarity threshold.

Vanrykel et al. in [52] present a graph building technique that processes a mobile traffic dataset in order to partition the network traces it contains by user. The idea is to exploit the sensitive identifiers that are typically present in the network traffic generated by mobile devices. The proposed methodology consists of the following steps:

1) Through the analysis of TCP timestamps, the packets that belong to the same app session (therefore to the same mobile user) are grouped together into a node. Alternatively, it is possible to partition the packets by TCP session.

2) For each node, the sensitive identifiers present in HTTP messages are extracted according to host-specific rules.

3) To cluster the nodes into components, each one representing the mobile traffic related to a specific mobile user, the following rules are iteratively applied to all nodes:

- If the node’s identifiers (or their hashed/encoded values) match the identifiers of an existing component, add the node to that component and merge their identifiers.
- If the node’s identifiers (or their hashed/encoded values) match the identifiers of multiple existing components, merge those components together, add the node to the resulting component, and merge their identifiers.
- If the node’s identifiers (or their hashed/encoded values) do not match the identifiers of any existing component, make the node a component on its own.

C. Machine Learning

Machine learning is the branch of artificial intelligence that studies algorithms that can be used to learn from and make predictions on data. Such algorithms are typically adopted to solve problems for which a traditional algorithmic solution (i.e., a finite sequence of instructions) is hard, if not impossible, to find. One of these problems is network traffic analysis.

In most of the works we survey, machine learning is effectively applied for mobile network traffic analysis. In the rest of this section, we describe the application of machine learning techniques to achieve a specific goal (see Section II): ad fraud detection (Section VI-C1), app identification (Section VI-C2), PII leakage detection (Section VI-C3), malware detection (Section VI-C4), operating system identification (Section VI-C5), tethering detection (Section VI-C6), traffic characterization (Section VI-C7), trajectory estimation (Section VI-C8), user action identification (Section VI-C9), user fingerprinting (Section VI-C10), and website fingerprinting (Section VI-C11).

1) Ad Fraud Detection: Crussell et al. in [30] present a system being able to automatically run Android apps in emulators and analyze their application-layer traffic in order to detect if they: (i) request ads while being in the background (i.e., ads are not displayed to the user); and (ii) click on ads without user interaction (i.e., false user clicks are simulated). The framework is based on supervised learning:

1) The HTTP and DNS traffic of each analyzed app is extracted from its network traces, then causally related HTTP requests are linked to form request trees.

2) For each request page (identified by the host and path names of its URL), all the related HTTP requests are aggregated. After that, the authors extract 33 features as follows:

- Ten features derive from query parameters:
  - For each query parameter, it is computed the ratio of distinct values found for that parameter over the total number of times the parameter appeared in a request, as well as the ratio of distinct values found for that parameter over the number of distinct apps. Each ratio is segmented into several intervals and the number of query parameters whose ratio is in each interval is counted. These counts contribute six features.
  - It is also computed the entropy of each query parameter. The entropy is considered high if it is greater than 216 bits, low otherwise. The number of query parameters that have, respectively, high and low entropy contribute two features.
  - The last two features are the average and the total number of query parameters.

- Sixteen features derive from the request trees. Such features are related to their structure (e.g., average height and depth of trees containing the page), number of children, their MIME types, and types of edge that connect the children to their parent.

- Seven features derive from HTTP headers (e.g., status codes, requests’ length, replies’ length).

3) The authors train random forest classifier is used to classify each request page as ad-related (ARQ) or not (NARQ). The performance of such classifier are 0.01% false-positive rate, 71.8% true-positive rate, and 85.9% class-weighted accuracy.

4) The ad request pages (i.e., the ARQ pages) and the HTTP request trees are leveraged to extract and verify ad impressions (i.e., displaying) and clicks.
The system is employed to analyze 130,339 Android apps crawled from nineteen different marketplaces, and 35,087 Android apps that likely contain malware provided by a security company. About 30% of apps with ads request to display an ad while running in the background, and 27 apps generate clicks without user interaction.

2) **App Identification:** Supervised learning is applied for app identification in [25], [55], [40], [43], [45], [58]. The methodology followed to build the app classifier is the same in all such works: (i) the network traffic of the selected mobile apps is captured; (ii) for each mobile app, feature vectors are extracted from its network traces and labeled with the name/type of that app; and (iii) the chosen classifier is trained on the labeled feature vectors. In Table [XVII] for each of the surveyed works in which machine learning is applied for app identification, we report the leveraged features and employed classifiers, as well as their effectiveness in identifying the apps in network traces. To complement the content of the table, we add that Le et al. in [36] describe only the groups to which the leveraged network-level features belong: packet length statistics; payload length statistics; inter-arrival time statistics; bursts timing; overall flow statistics; and TCP flags. Moreover, in the following we provide additional information about the reinforced-learning-based method that Taylor et al. propose in [58] to cope with the problem of ambiguous networks flows, i.e., network flows that are not useful in order to uniquely identify an app. Generated by third-party libraries (e.g., ad libraries) which can be embedded in different apps, such flows hinder the training of a classifier. To tackle the problem, the authors propose a method composed of four stages: (i) a preliminary classifier is trained using a preliminary training set; (ii) the preliminary classifier is evaluated using a preliminary testing set; (iii) samples which are wrongly labeled by the preliminary classifier are re-labeled as “ambiguous”; and (iv) a reinforced classifier is trained using the re-labeled dataset, including “ambiguous” as a new class.

3) **PII Leakage Detection:** Machine learning is applied for PII leakage detection in [24] and [48]. The authors of the former leverage hierarchical clustering (unsupervised learning), while the authors of the latter use a C4.5 decision tree (supervised learning).

Kuzuno and Tonami in [24] investigate the leakage of sensitive information due to ad libraries embedded into free Android apps. The proposed framework works as follows:

1) The HTTP traffic of the target mobile apps is captured.
2) The payloads of HTTP messages are inspected, and each message is labeled according to the fact that it contains sensitive information or not.
3) The HTTP messages containing sensitive information are clustered using hierarchical clustering. The following metrics are employed:
   - The HTTP message destination distance \(d_{\text{dst}}\), which is defined as:
     \[
     d_{\text{dst}}(p_x, p_y) = d_{\text{ip}}(p_x, p_y) + d_{\text{port}}(p_x, p_y) + d_{\text{host}}(p_x, p_y)
     \]  
     (VI-C3.1)

where \(p_n = \{ip_x, port_x, host_x\} \) with \(ip_n\) a destination IPv4 address, \(port_n\) a port number, \(host_n\) a HTTP host, and the distances are defined as:

\[
\begin{align*}
   d_{\text{ip}}(p_x, p_y) &= \text{lnmatch}(ip_x, ip_y)/32 \\
   d_{\text{port}}(p_x, p_y) &= \text{match}(port_x, port_y) \\
   d_{\text{host}}(p_x, p_y) &= \frac{\text{ed}(host_x, host_y)}{\max(\text{len}(host_x), \text{len}(host_y))}
\end{align*}
\]  

(VI-C3.2)

where \(\text{lnmatch()}\) returns the number of common upper bits in two IP addresses, \(\text{match()}\) returns 1 on matching ports and 0 otherwise, \(\text{ed()}\) returns an edit distance, \(\text{len()}\) returns the length of a character string, and \(\max()\) returns the greater of its two arguments. In particular, the values of distances \(d_{\text{ip}}\), \(d_{\text{port}}\) and \(d_{\text{host}}\) are within an interval \([0, 1]\).

- The HTTP message content distance \(d_{\text{header}}\), which is defined as:

\[
\begin{align*}
   d_{\text{header}}(p_x, p_y) &= d_{\text{rline}}(p_x, p_y) \\
   &\quad + d_{\text{cookie}}(p_x, p_y) + d_{\text{body}}(p_x, p_y)
\end{align*}
\]  

(VI-C3.3)

where \(p_n = \{\text{rline}_n, \text{cookie}_n, \text{body}_n\} \) with \(\text{rline}_n\) a request line, \(\text{cookie}_n\) a cookie, \(\text{body}_n\) a message body, and the distance is defined as:

\[
d_i(p_x, p_y) = \text{ncd}(i_x, i_y) \in [0, 1]
\]  

(VI-C3.4)

where \(i \in \{\text{rline}, \text{cookie}, \text{body}\}\) and \(\text{ncd}(k, z)\) is the normalized compression distance of the strings \(k\) and \(z\).

- Given \(C_x\) and \(C_y\) two clusters of HTTP messages, the linkage criterion is the following:

\[
d(C_x, C_y) = \frac{\sum_{p_x \in C_x, p_y \in C_y} d_{\text{msg}}(p_x, p_y)}{|C_x| \times |C_y|}
\]  

(VI-C3.5)

where:

\[
\begin{align*}
   d_{\text{msg}}(p_x, p_y) &= d_{\text{dst}}(p_x, p_y) \\
   &\quad + d_{\text{header}}(p_x, p_y)
\end{align*}
\]  

(VI-C3.6)

4) The conjunction signature set resulting from the clustering is employed to detect sensitive information leakage in mobile HTTP traffic.

The authors apply their solution to the network traffic of 1,188 free Android apps: 94% of HTTP messages containing sensitive information are correctly detected, with only 5% false negatives (i.e., sensitive HTTP messages that are not detected), and less than 3% false positives (i.e., non-sensitive HTTP messages incorrectly identified as sensitive).

Ren et al. in [48] focus on the mobile apps for Android, iOS, and Windows Phone. The presented framework is composed of three steps:

1) The collected network traffic (which consists of HTTP/HTTPS flows) is inspected looking for the PII related to the target mobile devices. Each flow is labeled according to the fact that it leaked PII or not.
TABLE XVII
THE SURVEYED WORKS IN WHICH MACHINE LEARNING IS APPLIED FOR APP IDENTIFICATION.

| Year | Paper                  | Features                                                                 | Classifier                      | Accuracy | Precision | Recall | F-measure |
|------|------------------------|---------------------------------------------------------------------------|---------------------------------|----------|-----------|--------|-----------|
| 2013 | Quazi et al. [25]      | Unspecified                                                               | C5.0 decision tree              | N/A      | N/A       | N/A    | 0.940     |
| 2015 | Le et al. [36]         | 84 network-level features                                                 | Linear support vector machine   | N/A      | N/A       | N/A    | 0.701     |
|      | Wang et al. [40]       | Twenty statistics on size and timing of 802.11 frames                     | Random forest                   | > 0.900  | N/A       | N/A    | N/A       |
| 2016 | Alan et al. [43]       | Burst sizes (rounded to the nearest 32 bytes) of the first 64 IP packets  | Based on Jaccard index          | 0.740    | N/A       | N/A    | N/A       |
|      |                        | Sizes of the first 64 IP packets (using the minus sign for incoming packets)| Gaussian naive Bayes            | 0.820    | N/A       | N/A    | N/A       |
|      |                        | Sizes of the first 64 IP packets (using the minus sign for incoming packets), modified by term frequency – inverse document frequency transformation and normalization | Multinomial naive Bayes         | 0.870    | N/A       | N/A    | N/A       |
|      | Mongkolhusamee et al. [45] | 35 graphlet- and 14 histogram-based features extracted from TCP/UDP network information (destination port) and statistics (total number of flows, total/per-flow number of bytes, and total/per-flow number of packets) | Random forest                   | N/A      | N/A       | N/A    | 0.960     |
| 2017 | Taylor et al. [58]     | 54 statistics computed on the IP packet sizes within TCP flows            | Random forest                   | ≤ 73% (worst) | 96% (best) | N/A    | N/A       |

2) In the feature extraction phase, a bag-of-words model is used, with the flows being the documents and the structured data being the words. More in detail, each flow is partitioned into words (using tokens), then it becomes a vector of binary values. In such vector, each word is set to 1 if it appears in the flow, otherwise it is set to 0.

3) For each destination domain (identified using the Host field of the HTTP header), the framework selects the features (i.e., the words) that are more suitable for classification. Finally, a C4.5 decision tree is trained on the feature vectors associated to that destination domain.

According to the authors, the classifier reaches an accuracy of 98.1%.

4) Malware Detection: Researchers effectively used machine learning techniques to detect mobile malware from network traffic. In particular, they applied both supervised learning [19, 21, 32, 46, 53, 54] and unsupervised learning [22].

a) Supervised Learning: In Table XVIII, for each of the surveyed works in which supervised learning is applied for malware detection, we report the leveraged features and employed classifiers, as well as their detection performance.

Regarding the work by Shabtai et al. in [19], the features reported in Table XVIII are only the ones related to network traffic. Moreover, the accuracy reported for each classifier is the average accuracy across all the experiments performed by the authors.

Regarding the work by Narudin et al. in [46], they rely on two datasets:

- The Priv dataset, which contains the network traffic of: (i) the top twenty free apps available in Google Play Store; and (ii) 30 new malware samples (from fourteen malware families) collected in 2013.

Regarding the work by Arora and Peddoju in [54], the authors present a feature selection algorithm to find the minimal set of features that achieves the best detection performance. Given a set of features \( \{ F_1, \ldots, F_n \} \), the proposed algorithm is composed of the following steps:

1) Rank the features according to different metrics. Each metric produces a different ranking. The authors uses the following metrics: (i) the information gain of a feature \( F \), which is the reduction of entropy after observing \( F \); and (ii) the chi-squared test, which express the difference between the expected and observed values.

2) For \( k = 1 \) up to \( n \):
   a) Extract the top-\( k \) features from each ranking, and keep only the ones that are present in all the rankings;
   b) Use the selected features to perform a classification using a naive Bayes classifier, and compute the achieved F-measure;
   c) If the F-measure computed above is greater than the one achieved in the previous steps, update the minimal set of features with the currently selected features.

b) Unsupervised Learning: The framework proposed by Wei et al. in [22] targets the Android platform, and works as follows:

1) A monitor collects DNS response messages.

2) The IP addresses within the answer and additional sections of the DNS response messages are mapped to geographical coordinates.

3) Independent Component Analysis (ICA) is used to compute the spatial uniform distribution of hosts (i.e., uniformity...
degree in the geographic distribution of hosts) and their spatial service relationship (which describes the relationship between a provider and a consumer by a service distance, and tends to be zero for a benign domain). Both of these metrics are leveraged to label mobile apps as benign or malicious.

Wei et al. evaluate their solution using a public Android malware dataset, as well as popular benign Android apps. Their classifier successfully identifies malicious apps with a nearly perfect accuracy.

5) Operating System Identification: Supervised learning techniques are also applied for operating system identification in [28], [29], [49], [57].

Chen et al. in [28] present a naive Bayes classifier that leverages the following binary features:

- $TTL = 128$ (Windows) and $TTL \neq 128$ (Windows, Android, or iOS), where $TTL$ is the Time-To-Live (TTL) field of the IP header.
- $ID_{mvr} < 0.05$ (mostly Windows, rarely Android), $ID_{mvr} \in [0.05, 0.40]$ (mostly Android, rarely Windows), and $ID_{mvr} > 0.40$ (mostly iOS, rarely Android), where $ID_{mvr}$ is monotonicity violation ratio of identification (ID) field in IP headers.
- $TS_{ratio} < 0.05$ (Windows) and $TS_{ratio} \geq 0.05$ (Android or iOS), where $TS_{ratio}$ is the ratio of segments with TCP timestamp option.
- $WS = 4$ (mostly Windows, rarely Android), $WS = 16$ (iOS), $WS = 64$ (mostly Android, rarely Windows), and $WS = 256$ (Windows), where $WS$ is the TCP window size scale option.

| Year | Paper | Features | Classifier | Accuracy | Precision | Recall | F-measure |
|------|-------|----------|------------|----------|-----------|--------|-----------|
| 2012 | Shabtai et al. [19] | Cellular/Wi-Fi sent/received bytes/packets | Bayesian networks | 0.654 | N/A | N/A | N/A |
|      |        |          | Decision tree | 0.792 | N/A | N/A | N/A |
|      |        |          | Histograms | 0.699 | N/A | N/A | N/A |
|      |        |          | $K$-means | 0.702 | N/A | N/A | N/A |
|      |        |          | Logistic regression | 0.813 | N/A | N/A | N/A |
|      |        |          | Naive Bayes | 0.868 | N/A | N/A | N/A |
| 2014 | Shabtai et al. [32] | Average and standard deviation of the number of sent/received packets, average and standard deviation of the number of sent/received bytes, and average session duration | Decision tree ($f_{148}$) | 0.916 | N/A | N/A | N/A |
|      |        |          | Decision forest | 0.967 | N/A | N/A | N/A |
| 2016 | Narudin et al. [46] | Source IP address, destination IP address, source port number, destination port number, frame length and number, HTTP request type, number of frames received by unique source/destination in the last $t$ seconds from the same destination/source, number of packets flowing from source to destination and vice versa | Bayesian network with feature selection on $D_{s1000}$ | N/A | 1.000 | 1.000 | 1.000 |
|      |        |          | Bayesian network on $Priv$ | N/A | 0.841 | 0.756 | 0.750 |
|      |        |          | Multi-layer perceptron with feature selection on $D_{s1000}$ | N/A | 0.887 | 0.838 | 0.880 |
|      |        |          | Multi-layer perceptron on $Priv$ | N/A | 0.880 | 0.840 | 0.839 |
|      |        |          | Decision tree ($f_{148}$) with feature selection on $D_{s1000}$ | N/A | 1.000 | 1.000 | 1.000 |
|      |        |          | Decision tree ($f_{148}$) on $Priv$ | N/A | 0.836 | 0.738 | 0.730 |
|      |        |          | $K$-nearest neighbors with feature selection on $D_{s1000}$ | N/A | 0.997 | 0.997 | 0.997 |
|      |        |          | $K$-nearest neighbors on $Priv$ | N/A | 0.884 | 0.846 | 0.845 |
|      |        |          | Random forest with feature selection on $D_{s1000}$ | N/A | 1.000 | 1.000 | 1.000 |
|      |        |          | Random forest on $Priv$ | N/A | 0.837 | 0.741 | 0.734 |
| 2017 | Wang et al. [53] | Per-TCP-flow sent/received bytes, sent/received packets, and average sent/received packet size | C4.5 decision tree | 98.16% TP rate, 5.14% FP rate | 99.65% TP rate, 1.84% FP rate |
|      | Arora et al. [54] | Minimal set: sent/received packets per second/flow, ratio of incoming to outgoing bytes, maximum/average packet size, minimum time interval between sent/received packets. Additional set: sent/received bytes per second/flow, ratio of incoming to outgoing packets, first sent/received packet size, maximum/average time interval between sent/received packets, average flow duration, ratio of number of connections to number of destination IPs. | Naive Bayes | Detection rate: 83.3% (minimal set of features), 87.25% (all features). |
• clock_{SD} \leq 3$ (mostly Android, rarely iOS and Windows) and \( \text{clock}_{SD} > 3 \) (iOS), where \( \text{clock}_{SD} \) is the standard deviation of a device clock frequency, estimated using packets coming from its IP address. The classifier is evaluated relying on Wi-Fi traces captured at an access point to which Android and iOS mobile devices, as well as Windows laptops, are connected. The authors report that their proposal identifies iOS with 1.0 precision and recall, while Android and Windows with 1.0 precision but 0.8 recall.

Coull and Dyer in [29] leverage the size of encrypted packets exchanged between a target iMessage user and Apple’s servers. The aim of this analysis is to determine whether the iMessage client is running on iOS or OS X. The authors use a binomial naive Bayes classifier with one class for each of the four possible (OS, direction) pairs, with direction indicating whether the packet is going to or coming from Apple’s servers. The classifier operates on a binary feature vector of (size, direction) pairs, where the value for a given feature is 1 if the corresponding pair is observed and 0 otherwise. Their proposal is able to achieve an accuracy of 100% accuracy within five packets.

The framework presented by Ruffing et al. in [49] combines together supervised learning and analysis of the frequency spectrum of packet timing. The proposed methodology is composed of two phases:

• Training phase:
  1) Each traffic trace, which is labeled with the operating system that generated it, is converted into a frequency spectrum.
  2) Frequency components are extracted from the frequency spectra generated in the previous step.
  3) A genetic algorithm is applied to separate the frequency components that are related to OS features from those that bring noise. The former are promoted features.

• Identification phase:
  1) A new traffic trace \( x \) is converted into a feature-extracted frequency spectrum \( F_x \).
  2) The identified operating system is provided by the following formula:

\[
\text{argmax}_{os \in \text{OS}} \frac{1}{n_{os}} \sum_{i=1}^{n_{os}} \text{corr}(F^x, F_i^{os})
\]

(\text{VI-C5.1})

where \( \text{OS} \) is the set of the considered mobile OSes, \( n_{os} \) is the number of feature-extracted frequency spectra of the mobile operating system \( os \), \( F_i^{os} \) is the \( i \)-th feature-extracted frequency spectrum of the mobile operating system \( os \), and \( \text{corr}(X,Y) \) is a function that computes the correlation between the frequency spectra \( X \) and \( Y \).

The authors evaluate their solution against network traffic captured within smartphones running the following operating systems: Android, iOS, Windows Phone, and Symbian. Defining the OS identification rate as the ratio of successful OS identifications over the number of attempts, the proposed framework reaches 70% OS identification rate for 30-seconds-long traces, and around 90% and above for traces lasting five minutes or more. Moreover, in case of heavy multitasking, the OS identification rate can reach 100% with only 30 seconds of network traffic. The authors also evaluate if their approach is suitable to discriminate different versions of the same OS, and they choose Android and iOS for such analysis. On fifteen-minutes-long traces from the Skype app, the OS identification rate can reach 98%, with misclassification rate below 10%. On traces of the same length but generated by the YouTube app, the OS identification rate is around 50%.

Malik et al. in [57] carry out OS identification by exploiting the inter-packet time of packets coming from the target mobile device. In particular, the authors focus on two types of packet: (i) the response to an ICMP packet sent to the target mobile device (active measurement); and (ii) an IP packet related to a video stream involving the target mobile device (passive measurement). A random forest classifier is trained and evaluated on the inter-packet times of three mobile devices running Android, iOS, and Windows Phone, respectively. The achieved accuracy is 75.2% for active measurement, and 73.6% for passive measurement.

6) Tethering Detection: Chen et al. in [28] apply supervised learning to detect if a target mobile device is tethering its Internet connection to other devices. The proposed probabilistic classifier leverages the following binary features:

- \( n_{OS} = 1 \) (no tethering) and \( n_{OS} > 1 \) (tethering), where \( n_{OS} \) is the number of operating systems identified from the packets coming from the same IP address (also the OS identification framework is based on machine learning, see Section [VI-C5] for details).
- \( n_{TTL} = 1 \) (no tethering with high probability) and \( n_{TTL} > 1 \) (tethering with high probability), where \( n_{TTL} \) is the number of distinct TTLs in the packets coming from the same IP address.
- \( ts_{mvr} \leq 0 \) and \( ts_{mvr} > 0 \), where \( ts_{mvr} \) is the violation ratio of the TCP timestamp monotonicity of the segments coming from the same device (the idea is to exploit the fact that segments generated by the same device tend to monotonically increase TCP timestamp values, whereas segments from different devices tend to have mixed TCP timestamp values).
- \( \text{clock}_{SD} \leq 35 \) and \( \text{clock}_{SD} > 35 \), where \( \text{clock}_{SD} \) is the standard deviation of the clock frequency estimated using the packets coming from the same IP address (a large standard deviation is likely due to tethering).
- \( \text{boot}_{SD} \leq 1455 \) and \( \text{boot}_{SD} > 1455 \), where \( \text{boot}_{SD} \) is the standard deviation of the boot time inferred from the TCP timestamp values in the segments coming from the same device (the idea is to exploit the fact that different devices have distinct boot times and distinct initial TCP timestamp values).

To evaluate their classifier, the authors use publicly available Wi-Fi traces collected at two conferences, as well as a one-week-long campus trace, and simulate tethering in each trace by randomly mixing packets from different IP addresses and modifying their source IP address to make them coming from the same IP address. On public traces, the classifier reaches 0.68-0.85 recall when the target precision is 0.95, and 0.78-0.89 recall when the target precision is 0.8; on the campus trace, the classifier reaches 0.86 precision, 0.74 recall, and 0.80 F-measure.
7) Traffic Characterization: Nayam et al. in [47] study the network behavior of 63 Android and 35 iOS free apps. To find similarities between the apps, the authors apply the following methodology:

1) The TCP and UDP traffic of each analyzed app is partitioned according to the type of domains to which it is related: (i) advertisement; (ii) tracking; (iii) popular services (e.g., Google, Facebook); and (iv) other domains.

2) For each app, the following attributes are computed: (i) total number of sessions; (ii) session rate for each type of domain; and (iii) percentage of sessions for each type of domain.

3) k-means clustering is applied to group together the apps that show a similar network behavior.

In Table XIX we report the resulting classification of the considered apps.

8) Trajectory Estimation: Musa and Eriksson in [18] present a system for converting the detections of the Wi-Fi probe requests periodically transmitted by a target mobile device into a highly likely spatiotemporal trajectory within the monitored area.

The trajectory estimation problem is formulated using a hidden Markov model (HMM): (i) each street of the covered area is partitioned into segments, and each segment represents a rectangular area in which a mobile device may be located; and (ii) a state of the HMM is assigned to each segment, and transition probabilities are used to model the behavior of mobile devices at intersections (i.e., go straight, turn left, or turn right). For each detection $det_i$ of the target mobile device and each state $s_i$ of the HMM, the emission probability $p(det_i|s_i)$, which represents the probability of making the detection $det_i$ if the current state is $s_i$, is computed. Finally, the Viterbi’s map-matching algorithm is applied to find the maximum-probability path, which is represented by a sequence of hidden states visited in the Markov model.

To evaluate their system, the authors set up three deployment and leverage GPS ground truth to measure the accuracy of the trajectory estimation: using monitors spaced over 400 meters apart, the mean error is under 70 meters.

9) User Action Identification: Machine learning is applied for user action identification in [27, 29, 37, 44, 50]. In this work, the authors leverage several techniques: in the field of unsupervised learning, agglomerative hierarchical clustering and k-means clustering; in the field of supervised learning, linear regression, naive Bayes, random forest, and support vector machine.

Watts et al. in [27] exploit the inter-packet time of responses to ICMP packets (i.e., pings) to infer the type of action that the target user is performing on her mobile device. In particular, the authors focus on three types of user action: (i) CPU intensive; (ii) I/O intensive; and (iii) non-CPU intensive. A random forest classifier is trained and evaluated on the inter-packet times of six Android apps, achieving a minimum 93% accuracy.

Coull and Dyer in [29] try to infer the language (among six possible choices: Chinese, English, French, German, Russian, and Spanish) and length of the messages exchanged between iMessage clients (on both iOS and OS X) and Apple’s servers.

For the language, a multinomial naive Bayes classifier is used with the count of each length/direction pair observed (direction indicates whether the data is going to or coming from Apple’s servers). Once we correctly identify that a message is sent/received by an iOS device, the language classification achieves more than 80% accuracy after 50 packets are observed. For the length, linear regression (with least squares estimation) is employed using the payload length as the explanatory variable and the message size as the dependent variable: the classification achieves an average error of 6.27 characters for text messages, and an absolute error of less than 10 bytes for attachment transfers.

Conti et al. in [44] combine unsupervised and supervised learning to fingerprint several user actions of popular Android apps. The proposed framework, whose classification performance is shown in Table XX, works as follows:

1) The network traffic generated by each user action is partitioned into flows (each flow is a time-ordered sequence of TCP segments exchanged during a single TCP session). Each flow is converted into three time series of packet sizes (with negative sizes for incoming traffic). One series is for incoming traffic, one is for outgoing traffic, and one combines traffic in both directions.

2) The flows are clustered using the agglomerative hierarchical clustering with the following linkage criterion:

$$d(u,v) = \sum_{1 \leq i \leq n} distance(u[i],v[j]) \quad \text{if} \quad |u[i] + v[j]| \leq 70 \text{ meters}$$

where $distance()$ is a distance function, and $u$ and $v$ are clusters of $n$ and $m$ elements, respectively. The distance function is defined as follows:

$$distance(f_i, f_j) = \sum_{k=1}^{n} w_k \times DTW(T_i^k, T_j^k)$$

where $f_i$ is a flow consisting of a set of $n$ time series $\{T_1^i, \ldots, T_n^i\}$, $w_k$ is a weight assigned to the $k^{th}$ time series, and $DTW(x, y)$ is the optimal warping path between the time series $x$ and $y$.

3) For every user action, each flow $f$ is assigned to the cluster that minimizes the distance between $f$ and the leader of the cluster (which is the flow that has the minimum overall distance from the other flows of the cluster). The $k^{th}$ feature indicates the number of flows that have been assigned to the $k^{th}$ cluster after the execution of that user action.

4) The final classification is performed using a random forest classifier.

The systems for user action identification developed by Park and Kim in [37] and Saltalformaggio et al. in [50] (whose classification performances are reported in Table XX) are similar to the one by Conti et al. [44]. However, there are a few differences:

- in [37], each flow is represented by a single time series including both incoming and outgoing traffic. As a consequence, the clustering is applied to time series (in [44], the
TABLE XIX
CLASSIFICATION OF THE APPS ACCORDING TO THEIR NETWORK BEHAVIOR (NAYAM ET AL. [17]).

| Cluster | Network behavior                                         | Classification       | Number of Apps |
|---------|----------------------------------------------------------|----------------------|----------------|
| 0       | Excessive ad-related traffic, and excessive number of sessions | Suspicious           | 42             |
| 1       | Excessive ad- and tracking-related traffic, and excessive number of sessions |                      |                |
| 2       | Excessive ad-related traffic, excessive traffic related to other domains, and excessive number of sessions |                      |                |
| 3       | Excessive tracking-related traffic, and excessive number of sessions |                      |                |
| 4       | Excessive ad-related traffic, but very low network activity |                      |                |
| 5       | High portion of traffic related to popular services, but very low use of them and very low network activity | Innocuous            | 29             |
| 6       | High use of popular services, but very low network activity |                      |                |
| 7       | High portion of traffic related to other domains, but very low tracking-related traffic and very low use of popular services | Potentially suspicious | 27             |
| 8       | High portion of traffic related to other domains, but very low use of them and very low portion of tracking-related traffic |                      |                |
| 9       | High portion of traffic related to other domains, but very low use of them |                      |                |

TABLE XX
CLASSIFICATION PERFORMANCE OF THE FRAMEWORKS THAT COMBINE CLUSTERING AND SUPERVISED LEARNING FOR USER ACTION IDENTIFICATION.

| Year | Paper                      | Precision | Recall | F-measure |
|------|----------------------------|-----------|--------|-----------|
| 2015 | Park et al. [37]           | 0.976     | 0.977  | 0.977     |
| 2016 | Conti et al. [44]          | 0.949     | 0.950  | 0.950     |
|      | Saltalamaggio et al. [50]  | 0.780     | 0.760  | N/A       |

authors consider sets of time series). Moreover, the distance function in the formula of the linkage criterion is simply DTW().

- in [50], the IP traffic is partitioned into server transactions, each containing the IP headers (ordered by time) of the packets exchanged with a specific remote host. The server transactions are converted into feature vectors by the following 26 features: (i) send/receive average inter-packet time; (ii) the ratio of the number of packets sent/received to/from the server over the total number of packets exchanged with the server; (iii) the ratio of the size of the data sent/received to/from the server over the size of the total data exchanged with the server; and (iv) the number of packets sent/received within each of ten size ranges, normalized by the total number of sent/received packets. The feature vectors are then clustered using k-means clustering (following an incremental approach to find a suitable value for k), and the final classification is performed using a multi-class support vector machine.

10) User Fingerprinting: Verde et al. in [33] present a framework for fingerprinting mobile users from NetFlow records. The proposed solution is based on supervised learning and hidden Markov model (HMM):

1) Feature vectors are extracted from the NetFlow records of the target user, and then partitioned into subsets. Each subset is related to a specific network service. For each service, several HMMs are created by varying the number of states and the subset of features, and setting up their parameters via the k-means algorithm. The HMMs are trained in parallel, and subsequently converted into binary classifiers using a probability threshold t (i.e., if the observation has a probability lower than t, it will be classified as 0, otherwise it will be classified as 1). Finally, the best performing HMM is selected for that network service.

2) Feature vectors are extracted from the NetFlow records captured at the monitored network, and subsequently partitioned according to the network service they belong to. Each feature vector is classified with the HMM corresponding to its service. For each time interval, the results are aggregated into a new record that contains, for each HMM h, the number of feature vectors recognized by h as belonging to the target user during the interval, times the weight of h (which is computed during the training phase). Finally, a machine learning classifier is used to determine whether, during each of the time intervals, the network traffic contains data transmissions from the target user. The framework supports the following classifiers: support vector machine, random forest, RIPPER, multilayer perceptron, and naive Bayes.

The system is evaluated as follows:

- Cross-validation is applied to the NetFlow records of the traffic generated by 26 different mobile users connecting to the Internet through the same Wi-Fi access point. The best performing implementation of the classifier (which is based on random forest) reaches 95% true-positive rate, 7% false-positive rate, 0.95 precision, 0.93 recall, and 0.94 F-measure.

- To profile them, five mobile users are induced to connect to a Wi-Fi access point under the control of the authors, who then
try to detect the presence of such users within a real-world large metropolitan Wi-Fi network. The implementation of the classifier based on random forest (the best performing) reaches over 90% true-positive rate, less than 8% false-positive rate, and over 0.9 precision and recall. It is worth to notice that the proposed solution is encryption-agnostic, since NetFlow records can be extracted even from encrypted traffic.

\textbf{11) Website Fingerprinting:} Spreitzer et al. in [51] fingerprint the websites visited by an Android user via the web browser of her mobile device, by leveraging the data-usage statistics of the browser app. The proposed framework is based on supervised learning:

1) Each considered website \( w_i \) is opened in the web browser of an Android device. At the same time, the TCP bytes transmitted and received by the browser app are sampled at a frequency \( f \) for a period of \( t \) seconds. The readings constitute a sample of the website \( w_i \). The sampling process stops after collecting \( n \) samples, which constitute the signature of the website \( w_i \). All the generated signatures are included in a database \( T \).

2) \( T \) is loaded into an unprivileged Android app that is installed on the target mobile device. When the user opens a website within the web browser of the device, the app samples the transmitted and received TCP bytes of the browser app, and builds a signature \( s \). For each signature \( s_i \) in \( T \), the similarity (which is a function based on the Jaccard index) between \( s \) and \( s_i \) is computed: the app returns the website corresponding to the signature \( s_i \) that maximizes the similarity with \( s \). The fingerprinting app can correctly infer 97% of 2,500 page visits out of a set of 500 monitored pages, and 95% of 500 page visits out of a set of 100 monitored pages when the traffic is routed through Tor by using the Orbot proxy in combination with the Orweb browser.

\textbf{VII. Conclusions}

In this paper, we surveyed the state of the art of the methods for analyzing the network traffic generated by mobile devices. In particular, we are the first that surveyed the works in which the mobile traffic is captured from alternative sources to cellular networks: Wi-Fi monitors and access points; wired networks; logging apps installed on mobile devices; and networks of mobile devices simulated via software. For each point of capturing, we described its characteristics, the number of mobile users that it monitors, as well as the issues related to the capturing process. Moreover, we observed that the most frequently used approach to capture mobile traffic is logging at either wired networks or mobile devices themselves.

We provide a systematic classification of the state of the art according to the goal of the analysis that targets the network traffic of mobile devices. In particular, we realized that most of the works focus on studying the features of the network traffic generated by mobile devices. Other popular goals are app and user action identification, usage study, and Personal Identifiable Information (PII) leakage and malware detection. While a lot of work has been done on such goals, promising topics, such as user fingerprinting and sociological inference, still offer much room for further investigation.

We also categorized the works on mobile traffic analysis according to the targeted mobile platforms. We observed that Android is not only the most popular mobile platform, but also the most targeted by the analysis methods (i.e., 42 out of 45 works which are not platform-independent). In fact, we demonstrated that the openness of the Android platform is a double-edged sword: on the one hand, it provides mobile users with a large number of apps that enable the most disparate functionalities; on the other hand, it helps malicious developers distribute malware, and more generally, apps that behave ambiguously with regard to the security of mobile devices and the privacy of mobile users.

Regarding the feasibility of the surveyed analyses in case of traffic encryption, we observed that 36 out of 56 works proposed methods that are able to carry out the analysis on traffic encrypted with SSL/TLS, while only 19 out of 36 can handle network traffic encrypted with IPsec. As in the traditional network traffic domain, the feasibility of the analysis on the encrypted traffic of mobile devices is bounded to the network layer in which the traffic has been captured. On one side, the analyses that focus on network traffic captured on a high layer (e.g., Application layer) have more fine-grained information available and generally achieve better results in terms of accuracy than on a lower layer (e.g., Network and Data-link layers). On the other side, the methods that focus on traffic on lower layers are more resilient to traffic encryption. As a practical example, a method that relies on Deep Packet Inspection (DPI) cannot analyze traffic encrypted with HTTPS, while a method that considers traffic from a Wi-Fi monitor (i.e., Data-link layer) can carry out the analysis even if the traffic is encrypted with IPsec.

Finally, we report that most of the surveyed works rely on machine learning techniques, with a prevalence of frameworks based on supervised learning, clustering, or a combination of both. For each framework, we explained its actual application as well as its achieved performance.

\textbf{Acknowledgment}

Mauro Conti is supported by a Marie Curie Fellowship funded by the European Commission (agreement PCIG11-GA-2012-321980). This work is also partially supported by the EU TagItSmart! Project (agreement H2020-ICT30-2015-688061), the EU-India REACH Project (agreement IC1+/2014/342-896), and by the projects “Physical-Layer Security for Wireless Communication” and “Content-Centric Networking: Security and Privacy Issues” funded by the University of Padua. This work is partially supported by the grant n. 2017-166478 (3696) from Cisco University Research Program Fund and Silicon Valley Community Foundation. This work is also partially funded by the project CNR-MOST/Taiwan 2016-17 “Verifiable Data Structure Streaming”. QianQian Li is supported by Fondazione Cassa di Risparmio di Padova e Rovigo.

\textbf{References}

[1] Statista, “Smartphone user penetration as percentage of total global population from 2014 to 2020,” November 2016.
[15] S. K. Baghel, K. Keshav, and V. R. Manepalli, “An investigation
on user experience,” in Proceedings of the 11th Annual International Conference on Mobile
Networking, ser. MobiCom ’12. New York, New York, USA: ACM, 2012, pp. 85–96.

[16] G. Maier, F. Schneider, and A. Feldmann, “A first look at mobile
hand-held device traffic,” in Proceedings of the 11th International Conference on Passive and Active Measurement, ser. PAM ’10.
A. Krishnamurthy and B. Plattner, Eds. Berlin, Heidelberg: Springer, 2010, pp. 161–170.

[17] G. Shadbolt, A. Ahmadi, C. Tossell, L. Zhong, and P. Kortum, “LiveLab: Measuring wireless networks and smartphone users in the field,” ACM SIGMETRICS Performance Evaluation Review, vol. 38, no. 3, pp. 15–20, December 2010.

[18] A. Finamore, M. Mellia, M. M. Munafò, R. Torres, and S. G. Rao, “YouTube everywhere: Impact of device and infrastructure synergies on user experience,” in Proceedings of the 2011 ACM SIGCOMM Internet Measurement Conference, ser. IMC ’11. New York, New York, USA: ACM, 2011, pp. 345–360.

[19] A. Gember, A. Anand, and A. Akella, “A comparative study of
handheld and non-handheld traffic in campus Wi-Fi networks,” in Proceedings of the 12th International Conference on Passive and Active Measurement, ser. PAM ’11. N. Spring and G. F. Riley, Eds. Berlin, Heidelberg: Springer, 2011, pp. 173–183.

[20] S.-W. Lee, J.-S. Park, H.-S. Lee, and M.-S. Kim, “A study on
smart-phone traffic analysis,” in Proceedings of the 13th Asia-Pacific Network Operations and Management Symposium, ser. APNOMS ’11. New York, New York, USA: IEEE Communications Society, 2011, pp. 177–183.

[21] A. Rao, Y.-s. Lim, C. Barakat, A. Legout, D. Towsley, and W. Dabbous, “Network characteristics of video streaming traffic,” in Proceedings of the 7th Conference on Emerging Networking Experiments and Technologies, ser. CoNEXT ’11. New York, New York, USA: ACM, 2011.

[22] S. K. Baghel, K. Keshav, and V. R. Manepalli, “An investigation into
traffic analysis for diverse data applications on smartphones,” in Proceedings of the 18th National Conference on Communications, ser. NCC ’12. New York, New York, USA: IEEE Communications Society, 2012, pp. 165–169.

[23] X. Chen, R. Jin, K. Suh, B. Wang, and W. Wei, “Network performance of smart mobile handhelds in a university campus WiFi network,” in Proceedings of the 2012 ACM SIGCOMM Internet Measurement Conference, ser. IMC ’12. New York, New York, USA: ACM, 2012, pp. 315–328.

[24] H.-S. Ham and M.-J. Choi, “Application-level traffic analysis of
smartphone users using embedded agents,” in Proceedings of the 14th Asia-Pacific Network Operations and Management Symposium, ser. APNOMS ’12. New York, New York, USA: IEEE Communications Society, 2012, pp. 244–247.

[25] A. B. M. Musa and J. Eriksson, “Tracking unmodified smartphones using Wi-Fi monitors,” in Proceedings of the 10th ACM Conference on Embedded Networked Sensor Systems, ser. SenSys ’12. New York, New York, USA: ACM, 2012, pp. 281–294.

[26] A. Shabtai, U. Kanonov, Y. Elovici, and Y. Weiss, “‘Andromaly’: A behavioral malware detection framework for Android devices,” Journal of Intelligent Information Systems, vol. 38, no. 1, pp. 161–190, February 2012.

[27] R. Stevens, C. Gibler, J. Crussell, J. Erickson, and H. Chen, “Investigating user privacy in android ad libraries,” in Proceedings of the 2012 Workshop on Mobile Security Technologies, ser. MoST ’12, 2012.

[28] X. Su, M. C. Chua, and G. Tan, “Smartphone dual defense protection framework: Detecting malicious applications in Android markets,” in Proceedings of the 9th International Conference on Mobile Ad-hoc and Sensor Networks, ser. MSN 2012. Oldwick, New Jersey, USA: IEEE Computer Society, 2012, pp. 153–160.

[29] T.-E. Wei, C.-H. Mao, A. B. Jeng, H.-M. Lee, H.-T. Wang, and D.-J. Wu, “Android malware detection via a latent network behavior analysis,” in Proceedings of the 11th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, ser. TrustCom ’12. Washington, District of Columbia, USA: IEEE Computer Society, 2012, pp. 1251–1258.

[30] M. V. Barbera, A. Epasto, A. Mei, V. C. Perta, and J. Stefa, “Signals from the crowd: Uncovering social relationships through smartphone probes,” in Proceedings of the 2013 ACM SIGCOMM Internet Measurement Conference, ser. IMC ’13. New York, New York, USA: ACM, 2013, pp. 265–276.

[31] H. Kuzuno and S. Tonami, “Signature generation for sensitive
information leakage in Android applications,” in Proceedings of the 29th IEEE International Conference on Data Engineering, ser. ICDE ’13. Washington, District of Columbia, USA: IEEE Computer Society, 2013, pp. 112–119.

[32] Z. A. Quazi, J. Lee, T. Jin, G. Bellala, M. Arndt, and G. Noubir, “Application-awareness in SDN,” in Proceedings of the 2013 Annual Conference of the ACM Special Interest Group on Data Communication, ser. SIGCOMM ’13. New York, New York, USA: ACM, 2013, pp. 487–488.

[33] A. Rao, A. Molavi Kakhki, A. Razaghpahan, A. Tang, S. Wang, J. Shan, P. Gill, A. Krishnamurthy, A. Legout, A. Mislove, and D. Choffnes, “Using the middle to meddle with mobile,” Northeastern University, Tech. Rep. NEU-CCS-2013-12-10, December 2013.

[34] L. Watkins, C. Corbett, B. Salazar, K. Fairbanks, and W. H. Robinson, “Using network traffic to remotely identify the type of applications executing on mobile devices,” in Proceedings of the 2nd Workshop on Mobile Security Technologies, ser. MoST ’13. Washington, District of Columbia, USA: IEEE Computer Society, 2013.

[35] Y.-C. Chen, Y. Liao, M. Baldi, S.-J. Lee, and L. Qiu, “OS fingerprinting and tethering detection in mobile networks,” in Proceedings of the 2014 ACM SIGCOMM Internet Measurement Conference, ser. IMC ’14. New York, New York, USA: ACM, 2014, pp. 173–180.

[36] S. E. Coul and K. P. Dyer, “Traffic analysis of encrypted messaging services: Apple iMessage and beyond,” ACM SIGCOMM Computer Communication Review, vol. 44, no. 5, pp. 5–11, October 2014.

[37] J. Crussell, R. Stevens, and H. Chen, “MadFraud: Investigating ad fraud in Android applications,” in Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services, ser. MobiSys ’14. New York, New York, USA: ACM, 2014, pp. T23–T34.

[38] M. Lindorfer, M. Neusgebwandner, L. Weichselbaum, Y. Fratantonio, V. van der Veen, and C. Platzer, “ANDRUBIS – 1,000,000 apps later: A view on current Android malware behaviors,” in Proceedings of the 3rd International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security, ser. BADGERS ’14. Washington, District of Columbia, USA: IEEE Computer Society, 2014, pp. 3–17.

[39] A. Shabtai, L. Tenenboim-Chekina, D. Minman, L. Rokach, B. Shapiro, and Y. Elovici, “Mobile malware detection through analysis of deviations in application network behavior,” Computers & Security, vol. 43, pp. 1–18, June 2014.

[40] N. V. Verde, G. Atieniese, E. Gabrielli, L. V. Mancini, and A. Spognardi, “No fingerprint’ed user left behind: Fingerprinting users behind NAT from NetFlow records alone,” in Proceedings of the 34th IEEE International Conference on Distributed Computing Systems, ser. ICDCS ’14. Washington, District of Columbia, USA: IEEE Computer Society, 2014, pp. 218–227.

[41] Z. Chen, H. Han, Q. Yan, B. Yang, L. Peng, L. Zhang, and J. Li, “A first look at Android malware ad libraries,” in Proceedings of 2015 IEEE Trustcom/BigDataSE/ISPA ’15. Washington, District of Columbia, USA: IEEE Computer Society, 2015, pp. 206–213.
[68] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel, “Website fingerprinting in onion routing based anonymization networks,” in Proceedings of the 10th Annual ACM Workshop on Privacy in the Electronic Society, ser. WPES ’11. New York, New York, USA: ACM, 2011, pp. 103–114.

[69] M. Conti, E. D. Santo, and R. Spolaor, “DELTA: data extraction and logging tool for android,” CoRR, vol. abs/1609.02769, 2016.

[70] Statista, “Mobile operating systems’ market share worldwide from January 2012 to June 2016,” November 2016. [Online]. Available: https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/ (visited on November 22, 2016).

[71] AppBrain, “Number of Android applications,” November 2016. [Online]. Available: http://www.appbrain.com/stats/number-of-android-apps (visited on November 23, 2016).

[72] Statista, “Number of available apps in the Apple App Store from July 2008 to June 2016,” November 2016. [Online]. Available: https://www.statista.com/statistics/263795/number-of-available-apps-in-the-apple-app-store/ (visited on November 23, 2016).

[73] L. A. Adamic and E. Adar, “Friends and neighbors on the web,” Social Networks, vol. 25, no. 3, pp. 211–230, July 2003.

[74] M. Müller, Information Retrieval for Music and Motion. Berlin, Heidelberg: Springer, 2007.