University of Padua

Department of Mathematics
Master degree in Computer Science

Can't you hear me knocking: Identification of user actions on Android apps via traffic analysis

Supervisor:
Dr. Mauro Conti
University of Padua, Italy

Co-Supervisor:
Dr. Nino Vincenzo Verde
Sapienza University of Rome, Italy

Candidate:
Riccardo Spolaor

External reviewer:
Dr. Mario Frank
European Patent Office, Germany

April 17th, 2014

"Can't you hear me knocking."

Rolling Stones – Can't you hear me knocking

"This is your here.
This is your now.
Let it be magical."

Ronnie James Dio – This is your life

Abstract

While smartphone usage become more and more pervasive, people start also asking to which extent such devices can be maliciously exploited as "tracking devices". The concern is not only related to an adversary taking physical or remote control of the device (e.g., via a malicious app), but also to what a passive adversary (without the above capabilities) can observe from the device communications. Work in this latter direction aimed, for example, at inferring the apps a user has installed on his device, or identifying the presence of a specific user within a network.

In this thesis, we move a step forward: we investigate to which extent it is feasible to identify the specific actions that a user is doing on his mobile device, by simply eavesdropping the device's network traffic. In particular, we aim at identifying user actions like browsing someone's profile on a social network, posting a message on a friend's wall, or sending an email.

We design a system that achieves this goal starting from encrypted TCP/IP packets: it works through identification of network flows and application of machine learning techniques. We did a complete implementation of this system and run a thorough set of experiments, which show that it can achieve accuracy and precision higher than 95%, for most of the considered actions.

This page is intentionally left blank

Acknowledgements

I would like to express the greatest gratitude to my advisor Dr. Mauro Conti for advising and encouraging me during all the development of this thesis. I want also to thank my co-supervisor Dr. Nino Vincenzo Verde for his suggestions and the time he dedicated to me. I would like to thank Dr. Fabio Aiolli, Dr. Michele Donini, and Dr. Mauro Scanagatta for their insightful comments. A special thanks also goes to my parents Renzo and Sonia for giving me the opportunity to achieve this goal.

Contents

Acknowledgements vii

Contents viii

List of Figures xi

List of Tables xiii

Acronyms xv

1 Introduction 1
- 1.1 Contributions . . . 2
- 1.2 Organization . . . 3

2 Related Work 5
- 2.1 Security and privacy on smartphones . . . 5
- 2.2 Application classification with traffic analysis . . . 7
- 2.3 Privacy attacks via traffic analysis . . . 8
- 2.4 Traffic analysis of mobile devices . . . 9

3 Machine learning and data mining background 11
- 3.1 Dynamic Time Warping . . . 11
- 3.2 Hierarchical Clustering . . . 12
- 3.3 Supervised Learning . . . 12
  - 3.3.1 Naive Bayes . . . 13
  - 3.3.2 Random forest . . . 13
- 3.4 Classification metrics . . . 14
  - 3.4.1 Confusion matrix . . . 14

4 Our framework 17
- 4.1 Network traffic pre-processing . . . 17
  - 4.1.1 Domain filtering . . . 18
  - 4.1.2 Packets filtering . . . 19
  - 4.1.3 Flow building . . . 19
- 4.2 Flows labeling . . . 20
- 4.3 Classification of user actions . . . 21

4.3.1 Flows clustering 21
4.3.2 User actions classification 22

5 Data acquisition and analysis **25**
5.1 Network traffic acquisition 25
5.1.1 Hardware and network configuration 25
5.1.2 Account and apps setup 26
5.2 Simulation of user actions using scripts 26
5.2.1 User actions sequence 28
5.2.2 Scripts execution 30
5.3 Domain study 30
5.3.1 Packets protocols 31
5.3.2 Traffic flows 31
5.3.3 User actions 33

6 Clustering and classification **39**
6.1 Unsupervised clustering 39
6.1.1 Statistical flow length distribution 39
6.1.2 Clustering parameters selection 41
6.2 Supervised classification 43
6.2.1 Training and test sets 43
6.2.2 Significant user actions 43
6.3 Classification algorithms comparison 43
6.4 Implementation tools 44

7 Classification performance **47**
7.1 Facebook 47
7.2 Gmail 48
7.3 Twitter 50

8 Conclusions **53**

A Script log files **55**

List of Figures

3.1 Example of a confusion matrix. 15

4.1 Flowchart for network traffic pre-processing procedure. 18
4.2 User action time intervals. 20
4.3 Flowchart for flow clustering procedure. 21

5.1 Coordinates for an object in Gmail layout. 29
5.2 Coordinates for an object in Twitter layout. 29
5.3 States flow on Facebook app due to user actions. 30
5.4 Representation of flows time series. 33
5.5 Network flows over time. 34
5.6 Presence of flows over time. 35
5.7 Flows related to an *post on wall* user action on Facebook. 36
5.8 Dendrogram of hierarchical clustering of flows for a user action. . . . 36

6.1 Statistical distribution of the length of complete flow time series. . . 40
6.2 Statistical distribution of the length of incoming flow time series. . . 40
6.3 Statistical distribution of the length of outgoing flow time series. . . 40
6.4 Representation of three different Gmail user actions. 42
6.5 Representation of three different Twitter user actions. 42
6.6 Classification accuracy using naive Bayes. 45
6.7 Classification accuracy using Random forest. 45

7.1 Classification accuracy of the Facebook user actions. 48
7.2 Facebook user actions confusion matrix. 49
7.3 Classification accuracy of the Gmail user actions. 50
7.4 Gmail user actions confusion matrix. 50
7.5 Classification accuracy of the Twitter user actions. 52
7.6 Twitter user actions confusion matrix. 52

List of Tables

5.1 Protocol presence among packets captured. 31
5.2 Example of time series generated by three flows. 32

6.1 Weights configurations for Gmail, Facebook and Twitter apps. . . . 41
6.2 Description of the Significant user actions for each app. 44

7.1 Classification results of Facebook user actions. 48
7.2 Classification results of Gmail user actions. 49
7.3 Classification results of Twitter user actions. 51

Acronyms

3G	**3**rd **G**eneration
ADB	**A**ndroid **D**ebug **B**ridge
FN	**F**alse **N**egative
FP	**F**alse **P**ositive
HMM	**H**idden **M**arkov **M**odels
HTTP	**H**yper**T**ext **P**rotocol
HTTPS	**H**yper**T**ext **P**rotocol **S**ecure
IP	**I**nternet **P**rotocol
MAC	**M**edia **A**ccess **C**ontrol
ML	**M**achine **L**earning
NB	**N**aive **B**ayes
OS	**O**perative **S**ystem
OSI	**O**pen **S**ystems **I**nterconnection
OSN	**O**n-line **S**ocial **N**etwork
P2P	**P**eer to **P**eer
QoS	**Q**uality **o**f **S**ervice
RTT	**R**ound-**T**rip **T**ime
SDK	**S**oftware **D**evelopment **K**it
SSH	**S**ecure **SH**ell
SSL	**S**ecure **S**ockets **L**ayer
SVM	**S**upport **V**ector **M**achine
TAA	**T**ime **A**fter a user **A**ction
TCP	**T**ransmission **C**ontrol **P**rotocol
TLS	**T**ransport **L**ayer **S**ecurity
TP	**T**rue **P**ositive
UMTS	**U**niversal **M**obile **T**elecommunications **S**ystem
VOIP	**V**oice **O**ver **IP**

Chapter 1

Introduction

Nowadays smartphones are widely used and pervasive devices. People continuously carry those devices with them and use them more and more for daily communication activities, including not only voice calls and SMS but also emails and social network interaction. In the last years, several concerns have been raised about the capabilities of those portable devices to invade the privacy of the users and actually becoming "tracking devices". One aspect is concerned with the possibility of continuously localize an individual [1, 2]. Another relevant aspect is related to the fact that malicious apps can go even a step further in tracing and spying on someone life. For example, a malicious app that has access to the microphone and networking capabilities, could in principle continuously eavesdrop the audio and send it over the Internet to an adversary [3].

Even when the adversary has no actual control of the phone (either physical control or remote via malicious apps) other attacks in the same directions are possible to violate the privacy of the communications. If the network traffic is not encrypted, the task of the eavesdropper is simple, since he can analyze the payload and read the content of each packet. However, many mobile apps use the Secure Sockets Layer (SSL)—and its successor Transport Layer Security (TLS)—as a building block for encrypted communications. In a typical SSL/TLS usage scenario, a server is configured with a certificate containing a public key as well as a matching private key. As part of the handshake between an SSL/TLS client and server, the server proves it has the private key signing its certificate with public-key cryptography. Unfortunately there is often a gap between theory and practice, e.g., leveraging the SSL vulnerabilities of smartphone apps [4, 5] one might run a SSL man-in-the-middle attack to compromise the confidentiality of communications.

We believe that while people become more familiar with mobile technologies and their related privacy threats (also thanks to the attention raised by media, e.g., see the recent attention on NSA for supposedly eavesdropping foreign governments leader such as Angela Merkel [2]), users start adopting some good practices that better adapt to their privacy feeling and understanding. For examples, solutions to identify and isolate malware running on smartphones [6, 7, 8] as well as to protect against attacks coming from the network [9, 10] might significantly reduce current threats to user privacy.

Unfortunately, we believe that even adopting such good practices would not close the door to malicious adversaries willing to trace people. In fact, the wireless and pervasive nature of mobile devices would still leave many practical options for adversarial tracing. In particular, even when such solutions are in place, the adversary can still infer a significant amount of information from the properly encrypted traffic. For example, work leveraging analysis of encrypted traffic already highlighted the possibility of understanding the apps a user has installed on his device [11], or identify the presence of a specific user within a network [12].

This work focuses on understanding whether the user profiling made through analyzing encrypted traffic can be pushed up to understand exactly what actions the user is doing on his phone: as concrete examples, we aim at identifying user actions such as sending an email, receiving an email, browsing someone profile in a social network, rather than "tagging" someone in a picture. The underlying issue we leverage in our work is that SSL and TLS protect the content of a packet, while they do not prevent the detection of networks packets patterns that instead may reveal some information about the user behavior.

1.1 Contributions

In this thesis, we propose a framework to infer which particular actions the user executed on some app installed on his mobile-phone, only looking at the network traffic that the phone generates. In particular, we assume the traffic is encrypted and the attacker eavesdrops (without modifying them) the messages exchanged between the user's device and the web services that he uses.

Our framework analyzes the network communications and leverages information available in TCP/IP packets (like IP addresses and ports), together with other information like the size, direction (incoming/outgoing), and timing. Using an approach based on machine learning, each app that is of interest is analyzed independently.

To set up our system, for each app we first pre-process a dataset of network packets labeled with the user actions that originated them, we cluster them in flow typologies that represent recurrent network flows, and finally we analyze them in order to create a training set that will be used to feed a classifier. The trained classifier will be then able to classify new traffic traces that have never been seen before.

We fully implemented our system, and we run a thorough set of experiments to evaluate our solution considering three very popular apps: Facebook, Gmail, and Twitter. The results shows that it can achieve accuracy and precision higher than 95%, for most of the considered actions done by the user with those apps.

1.2 Organization

The remainder of this thesis is organized as follows. In Chapter 2, we revise the state of the art around our research topic. We introduce in Chapter 3 some background knowledge, used in our work, on machine learning and data mining tools. In Chapter 4, we present an overview of our framework and its components. We report in Chapter 5 the methodology followed for data acquisition and the preliminary studies carried out on that data. Following, we discuss about implementation details for clustering and classification algorithms in Chapter 6. In Chapter 7, we present the evaluation of our solution for identifying user actions. Finally, in Chapter 8 we draw some conclusions and discuss about possible countermeasures against the attack.

Chapter 2

Related Work

Our main claim in this thesis is that network traffic analysis and machine learning can be used to infer private information about the user, i.e., the actions that he executes with his mobile phone, even thought the traffic is encrypted. To position our contribution with respect to the state of the art, in this chapter we survey the works that belong to research areas that focus on similar issues: *security and privacy on smartphones*, *application classification with traffic analysis*, *privacy attacks via traffic analysis* and *traffic analysis of mobile devices*.

2.1 Security and privacy on smartphones

Privacy is an important matter perceived by users of smartphones, even more than using a laptop [13]. Malwares are a serious threat to security on smartphone [14], because they can cause device malfunctioning and user personal information disclosure, like user position, contacts, health condition, etc.

Some works propose apps profiling frameworks to detect malicious behavior or potential privacy-related information exposure [15, 16, 17]. In [15], Eder et al. show an extendable framework named ANANAS, that analyze static and dynamic behaviors of Android apps. With dynamic behavior, the authors mean that apps are subject to the activity of a user, while static behavior stands for inactive status. Using these distinct analysis approaches, authors are able to evaluate how user activity can impact on apps behavior, marking a significant difference between static and dynamic behaviors. From these analysis ANANAS collect apps behavior samples to compare with Android Malware Genome Project, and identify malicious intents

or personal information leaks. This framework supports an abstraction layer for simple user interaction and phone event simulation, similar to the scripts we use in this thesis to collect data. In [17], Yang et al. present a framework to validate new apps, that helps human analysts to determine if a data transmission is intended by the user. Indeed, authors consider that sometime users intend to transmit personal information using their smartphones, and this has not to be considered as a privacy leak. So it is reasonable to consider as malicious only traffic not related to any user action (intent). Among these frameworks, Wei et al. [18] propose ProfileDroid. Differently from previous solutions, app profile is done from four different points of view (layer): static, user interaction, OS and network.

In Android OS, malicious apps could obtain the access to device resources through given permissions. But in most cases, users do not understand what these permissions really mean [19]. Possible countermeasures to this problem consist to replace sensitive data with a shadowed copy [20] and let apps believe they are working on real data, or use taint analysis [21] to track and understand how apps use these data. Another possible approach to this problem is MockDroid [22], a modified version of Android OS, that gives to apps a fake access to resources. With this approach, users are able to select a trade off between more privacy and a reduction of functionality, customizing the permissions given to apps (true or fake).

Another scenario of personal data leak is when an unauthorized user tries to physically access a device. To mitigate this threat, the device has to be able to verify the user identity, using measures based on a secret (e.g., PIN, password) or biometrics (e.g., face recognition, fingerprint). In particular, researchers leverage machine learning techniques to authenticate a user using biometrics. For example, Ho et al. in [23] show how it is possible to verify the user identity using biometrics related to gait. Regarding this topic, Conti et al. propose a measure to authenticate a user from the way he answers to a phone call [24]. Monitoring accelerometer and orientation sensors, the authors are able to produce a unique fingerprint of a user tracing the movement of his hand. More recently, Majdanik et al. in [25] use these sensors to authenticate a user from his movements while he is typing on device keyboard.

2.2 Application classification with traffic analysis

In recent years, researchers put a significant effort in traffic analysis, aiming to identify which application produces a network flow. In the works we report in this section, authors mean with the term "application" the last layer in OSI model. Particular interest focus on P2P traffic recognition (emule, bittorrent), but also in application level protocols like HTTP, POP3, SSH and so on. Traffic analysis can be done in several ways exploiting machine learning algorithms on different network traffic features. About this topic, a very useful work is *Internet Traffic Classification Demystified* [26], that help to understand which features of network flows are the most significant on traffic analysis. In [27], Kim et al. report an overview on best practices for application classification via traffic analysis. In this work, authors compare machine learning algorithms, depending from the approaches used: ports-based, host-behavior-based, and flow-features-based.

Works on traffic analysis can be distinguished from each other according to the machine learning methods they use:

- Naive Bayes [28, 29];
- Hidden Markov Models (HMM) [12, 30, 31, 32];
- unsupervised clustering and supervised classification [33, 34, 35];
- Support Vector Machines (SVM) [33, 36, 37];
- custom classification algorithms [38, 39, 40].

Many methods proposed in these works can be apply on encrypted TCP/IP traffic, because they consider only statistics of network flows (without access to packet payloads). An example of custom traffic classifier is proposed by Crotti et al. in [39], where authors build a fingerprint for a protocol using statistics about packets size, inter-arrival time and order. So they propose an ad-hoc classification algorithm to classify these fingerprints. In [38], Karagiannis et al. propose a framework that use "multi-level" analysis on TCP traffic. This framework, the authors named BLINC, considers network traffic from three different points of view: hosts popularity (social level), hosts role in the network (client/server level), and traffic flow features (application level). Finally, the authors propose a custom classifier that combines features from each level.

In this thesis, we use a method known as early traffic analysis, that consist in consider only a limited number of packets of a traffic flow. In [33, 36], Sena et al. describes two on-line methods to classify encrypted traffic. Starting from a payload traffic analysis as ground truth, they compare centroid clustering and SVM, applying them on statistical flow analysis. The authors do not consider a whole flow, but only the first N packets (early) in both directions. Although in their flow representation, information about packets sequence is not take in concern. Early flows analysis is also used in [30] for application recognition. First studying early TCP connection features, then evaluating performance of HMM on this domain. Some others, like [29], do the same classification but timely and continuously, oriented to QoS management. Their traffic classifier uses C4.5 Decision Tree and Naive Bayes algorithms on statistics of sub-flows (a limited number of packets taken at any point on a traffic flow). They are able to identify interactive traffic (on-line gaming and VOIP) among TCP/IP traffic.

In our framework, we use a classification in two steps: first we regroup flows in clusters according to their similarities, then we use those clusters as features to classify user actions. A similar approach, but with a different purpose, is used in [35] on HTTP traffic and in [34] on TCP traffic.

2.3 Privacy attacks via traffic analysis

In the literature, several works propose to track user activities on the web analyzing unencrypted HTTP requests and responses [41, 42, 43]. With this analysis it is possible to understand user actions inferring interests and habits. In particular, Schneider et al. [42] studied how users interact with Online Social Network (OSN) analyzing anonymized HTTP header. With their method, It is possible follow user activity on OSN web pages, inferring user actions. However, in recent years, websites and social networks started to use SSL/TLS encryption protocol, both for web and mobile services [44, 45]. This means that communications between endpoints are encrypted and this type of analysis cannot be performed anymore.

Different works survey possible attacks that can be performed using traffic analysis assuming a very strong adversary (e.g., a national security agency) which is able to observe all communication links [46, 47]. In [48], Liberatore et al. evaluate the effectiveness of two traffic analysis techniques based on naive Bayes and on Jaccard's coefficient for identifying encrypted HTTP streams. Such an attack is outperformed

by Hermann et al. in [49], where they present a method that applies common text mining techniques to the normalized frequency distribution of observable IP packet sizes, obtaining a classifier that correctly identifies up to 97% of requests. The technique is further refined in [50], where Panchenko et al. present a support vector machine classifier that is able to correctly identify web-pages, even when the victim use both encryption and anonymization networks such as Tor.

Traffic analysis is applied not only to HTTP but also to other protocols. For example, Song et al. [32] prove that SSH is not secure. In particular, they show that even very simple statistical techniques suffice to reveal sensitive information such as login passwords. More importantly, the authors show that using more advanced statistical techniques on timing information collected from the network, the eavesdropper can also learn significant information about what users type in SSH sessions. Developing a Hidden Markov Model, they are able to predict key sequences from the inter-keystroke timings. SSH is not the only protocol that has been target of such attacks. In fact, another example is Voice Over IP (VoIP). In particular, in [31], Wright et al. show how the lengths of encrypted VoIP packets can be used to identify spoken phrases of a variable bit rate encoded call. Their work indicates that a profile Hidden Markov Model trained using speaker- and phrase-independent data can detect the presence of some phrases within encrypted VoIP calls with recall and precision exceeding 90%.

Traffic analysis can also be apply on data link layer, analyzing MAC (Media Access Control) packets exchanged. For example, Zhang et al. [37] profile applications behavior using traffic analysis, they are able to infer which kind of activity a user is doing by eavesdropping MAC packets transmissions for a few minutes.

2.4 Traffic analysis of mobile devices

Focusing on mobile devices, traffic analysis is successfully used to detect information leaks [21], to profile users from their set of installed apps [11], and to produce the fingerprint of an app from its HTTP traffic [51, 52]. Traffic analysis is also used to understand network traffic characteristics, with particular attention on energy saving [53].

Stober et al. [11] show that it is possible to identify the set of apps installed on an Android device, by eavesdropping the 3G/UMTS traffic that those apps generate. Similarly, Tongaonkar et al. [51] introduce an automatic app profiler that creates

the network fingerprint of an Android app in order to re-identify its HTTP traffic. To increase the accuracy of the profiler, this work is extended considering in-app advertisements traffic [52]. Unfortunately, these methods could not be applied on Android apps, including the ones considered in our work. To prove this statement, we observe that network traffic produced by Facebook, Gmail and Twitter involves encrypted connections. In these connections, exchanged packets' protocol is TLSv1 for over the 95% of them (more details are reported in Section 5.3.1). Furthermore, these apps do not contain in-app advertisements, regardless they are free. None of the works we mention in this section aim at inferring the actions performed by the user over its mobile phone, which is the goal of this thesis.

Chapter 3

Machine learning and data mining background

In this chapter, we introduce several concepts and tools about machine learning and data mining used in this thesis.

3.1 Dynamic Time Warping

Dynamic Time Warping (DTW) [54] is a useful method to find alignments between two time-dependent sequences (also referred as time series) which may vary in time or speed. This method is also used to measure the distance or similarity between time series. Let us consider two sequences that represent two discrete signals:

- $X = (x_1, \ldots, x_n)$ of length $N \in \mathbb{N}$;
- $Y = (y_1, \ldots, y_m)$ of length $M \in \mathbb{N}$.

DTW uses a local distance measure $c : \mathbb{R} \times \mathbb{R} \to \mathbb{R}_{\geq 0}$ to calculate a cost matrix $C \in \mathbb{R}^{N \times M}$. The cell $C_{i,j}$ of this matrix reports the distance between x_i and y_j. The goal is to find an alignment between X and Y having minimal overall distance. Intuitively, such an optimal alignment runs along a "valley" of low cost cells within the cost matrix C. More formally, a *warping path* is defined as a sequences $p = (p_1, \ldots, p_L)$ with $p_l = (n_l, m_l) \in [1 : N] \times [1 : M]$, $l \in [1 : L]$ satisfying the following three conditions:

1. Boundary condition: $p_1 = (1, 1)$ and $p_L = (N, M)$;
2. Monotonicity condition: $n_1 \leq n_2 \leq \ldots \leq n_M$ and $m_1 \leq m_2 \leq \ldots \leq m_L$;
3. Step size condition: $p_{l+1} - p_l = \{(0, 1), (1, 0), (1, 1)\}$ for $l \in [1 : L - 1]$.

The total cost of a warping path is calculated as the sum of all the local distances of its elements. An *optimal warping path* is a warping path p^* having minimal total cost among all possible working paths. The total cost of an *optimal warping path* is also used as a distance measure between two sequences X and Y. In this thesis, we will indicate the cost of an *optimal warping path* with $DTW(X, Y)$.

3.2 Hierarchical Clustering

Hierarchical clustering is a cluster analysis method which seeks to build a hierarchy of clusters. This clustering method has the distinct advantage that any valid measure of distance can be used. In fact, the observations themselves are not required: all that is used is a matrix of distances.

In the following, we will use a type of hierarchical clustering that is called agglomerative: each observation starts in its own cluster, and pairs of clusters are merged as one moves up the hierarchy. In order to decide which clusters should be combined, a metric (a measure of distance between pairs of observations) and a linkage criterion are required. Since we apply clustering to time-dependent sequences, we use the total cost of an *optimal warping path* as distance metric. As for the linkage criterion, that determines the distance between sets of observations as a function of the pairwise distances between observations, we use the average distance, that is defined as:

$$d(u, v) = \sum_{\substack{1 \leq i \leq n \\ 1 \leq j \leq m}} \frac{d(u[i], v[j])}{|u| * |v|},$$

where $d()$ is a distance function, and u and v are two clusters of n and m elements, respectively. More details about hierarchical clustering can be found in [55].

3.3 Supervised Learning

Supervised machine learning algorithms learns from labeled instances or examples, which are collected in the past and represent past experiences in some real-world

applications. They produce an inferred model, which can be then used for mapping or classifying new instances. An optimal scenario will allow for the algorithm to correctly determine the class labels for unseen instances.

In this thesis, we use a naive Bayes classifier [56] and an ensemble classifier that is called Random forest [57].

3.3.1 Naive Bayes

Naive Bayes classifies are supervised learning algorithms based on theorem of Bayes, with the assumption of independence between every pair of features. Given a class y and a vector of features x, theorem of Bayes with the "naive" assumption states the following relationship:

$$P(y \mid x_1, \ldots, x_n) = \frac{P(y) \prod_{i=1}^{n} P(x_i \mid y)}{P(x_1, \ldots, x_n)}.$$

Naive Bayes classifiers differ mainly by the assumptions they make regarding the distribution of $P(x_i \mid y)$.

In our analysis we use the *Gaussian* distribution, where the likelihood of the features is assumed to be Gaussian:

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\pi\sigma_y^2}\right).$$

Where the parameters σ_y and μ_y are estimated using maximum likelihood. More details about naive Bayes classifiers can be found in [56].

3.3.2 Random forest

The main principle behind ensemble methods is that a group of "weak learners" can be combined together to form a "strong learner". Random forest leverages a standard machine learning technique called "decision tree", which, in ensemble terms, corresponds to the weak learner. In practice, it combines together the results of several decision trees trained with different portions of the training dataset and different subsets of features. More details about the Random forest classifier can be found in [57].

3.4 Classification metrics

In machine learning and data mining, there are metrics to measure the accuracy of a trained classifier using its predictions on a test set. Knowing the true classes of examples in test set, it is possible to compare them with predictions and identify the number of true positives (TP), false positives (FP) and false negatives (FN) for every class.

Using these values it is possible to calculate:

- $precision = \frac{TP}{TP+FP}$, evaluates for a class the number correct classified examples, over the total number of examples classified as belonging to that class;
- $recall = \frac{TP}{TP+FN}$, evaluates for a class the number correct classified examples, over the number of examples that really belongs to that class.

To measure the accuracy of the classification, F-measure (also knows as F1 score) consider both precision and recall with equal weight:

$$F_1 = 2 * \frac{precision * recall}{precision + recall} = \frac{2 * TP}{2 * TP + FP + FN}.$$

3.4.1 Confusion matrix

A graphical instrument for evaluate the performance of classification is the confusion matrix. In this matrix, every true class is reported in a row, re-scaled between 0 to 1 over total number of examples for that class. In other hand, on columns there are classes prediction. An ideal classifier performance report a confusion matrix with values equal to 1 on the diagonal, but 0 on other elements. For each row, it is reported how many examples (over total examples for that class) are predicted as belonging to the classes in columns. So in a row of a confusion matrix, the correspondent element on diagonal coincide with the recall for that class, while FN ratio on other elements.

In Figure 3.1, we report an example of confusion matrix. Considering the row related to *Class_B*, we observe that the 80% of examples for that class are classified correctly, while the 5% and the 15% of them are wrongly classified as *Class_D* and *Class_E* respectively. In the other hand, observing the column related to *Class_A*, the recall for that class is 1.0, so the 100% of the examples are classified correctly.

Despite this, some examples of *Class_C* and *Class_E* (4% and 7% respectively) are wrongly predicted as *Class_A*, lowering the precision for that class.

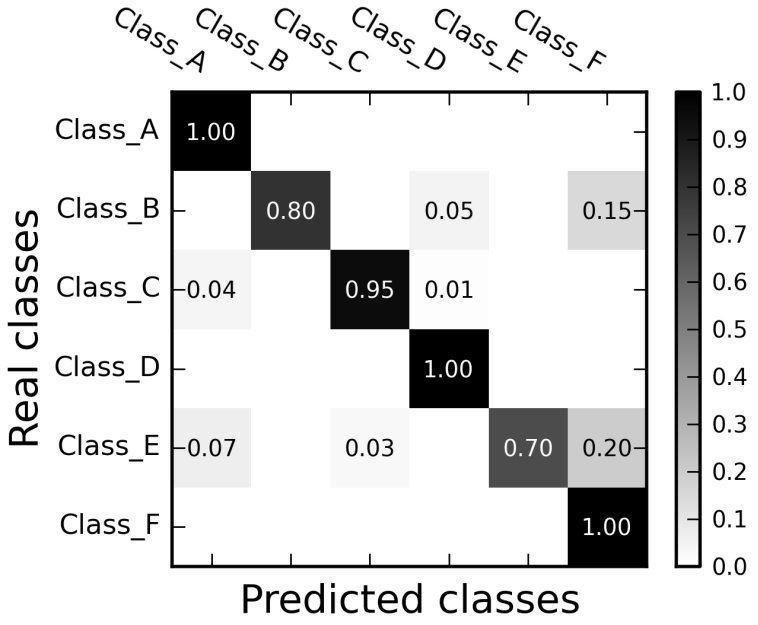

FIGURE 3.1: An example of a graphical representation for a confusion matrix.

Chapter 4

Our framework

In this chapter, we describe the components of our framework. In particular, we introduce the concept of flow and we describe the pre-processing steps that allow us to model the network traffic. Then, we report the methodology followed to assign flows to the user action that generates them. Finally, we describe the procedure to build training and test dataset, and the algorithms used to classify user actions.

4.1 Network traffic pre-processing

Mobile apps generally rely on SSL/TLS to securely communicate with peers. These protocols are built on the top of the TCP/IP suite. The TCP layer receives encrypted data from the above layer, it divides data into chunks, and adds a TCP header creating a TCP segment. The TCP segment is then encapsulated into an Internet Protocol (IP) datagram, and exchanged with peers. Since TCP packets do not include a session identifier, both endpoints identify a TCP session using the client's IP address and the port number. A packet is a sequence of bytes, and consists of a header followed by a payload. The header describes the packet's source, destination and control information. The payload contains the data is transmitting. In Section 5.1, we describe in details the methodology we follow to acquire these packets.

A fundamental entity considered in this thesis is the *flow*: with this term we indicate a time ordered sequence of packets exchanged between two peers during a single TCP session. In the following, we describe the procedure used to pre-process

network traffic and flow labeling, we also report a scheme for this procedure, using the flowchart in Figure 4.1.

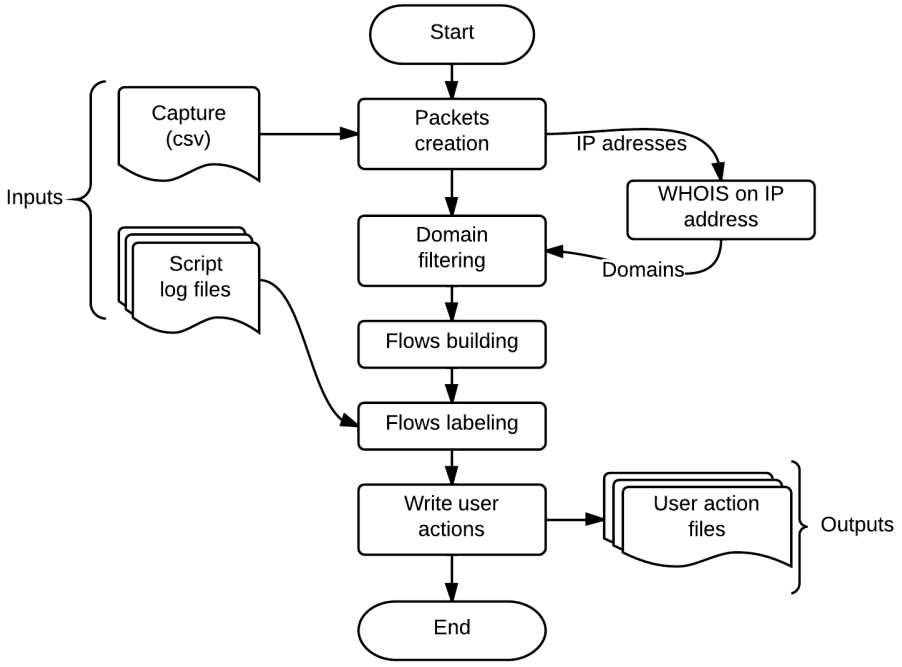

FIGURE 4.1: Flowchart for network traffic pre-processing and labeling procedure.

Before generating flows from each TCP session, a few pre-processing steps have to be performed:

1. we apply a domain filtering to select only packets belonging to the analyzed app;
2. we filter the remaining packets, in order to delete the ones of them that may degrade the precision of our approach (i.e., we filter out ACK and retransmitted packets);
3. we generate flows limiting the duration of a TCP session.

4.1.1 Domain filtering

Since we analyze each app independently, we need to make sure that traffic generated by apps other than the considered one (or traffic generated by the OS) do not interfere with the analysis. In order to univocally identify the app that generates a particular flow, we use the destination IP addresses, and the WHOIS protocol to

perform a series of look up, inferring information about the owner of the destination IP address. When the owner of the IP address is clearly related to the app which is subject of the analysis, we take the flow into consideration for further analysis. Since several apps also use third parties services (such as Akamai, or Amazon, as reported in [18]), we take also these flows into consideration. In all the other cases, we simply discard the flow.

4.1.2 Packets filtering

Due to network congestion, traffic load balancing, or other unpredictable network behavior, IP packets can be lost, duplicated, or delivered out of order. TCP detects these problems, hence requesting retransmission of lost data, and reordering out-of-order data. It comes out that several TCP packets that do not carry data, may hinder the analysis process. In the data exchange phase, for example, the receiver sends a packet with the ACK flag set to notify the correct reception of a chunk of data. These ACK packets are transmitted in asynchronous mode so they are affected from many factors related to round trip time of the connection link. The order of the received packets may hinder the evaluation of the similarity between two flows. For this reason, we filter out all packets retransmissions, as well as packets marked with the ACK flag. We filter out also other packets that do not bring any additional information helpful in characterize flows. In particular, we filter out the three way handshake executed to open a TCP connection, and the packets exchanged to close it.

4.1.3 Flow building

Consecutive packets, with the same couples of IP addresses and port for source and destination, are aggregated in flows. We model each flow as a set of time series: (i) a time series is obtained considering the bytes transported by incoming packets only; (ii) another one is obtained considering bytes transported by outgoing packets only; (iii) a third one is obtained combining bytes transported by both incoming and outgoing packets. Additional time series may be obtained, for example, considering other parameters such as the time-gap between different packets. However, in our analysis we only use the first three types mentioned above. In Section 5.3.2, we report more details about flow time series representation we use in our framework.

To limit the duration of a flow we use a mechanism based on a timeout. We consider a flow as terminated when there is not any new packet since a timeout of 4.5 seconds. Indeed, it has been proved experimentally that 95% of all packets arrive at most 4.43 seconds after their predecessors [11].

4.2 Flows labeling

After pre-processing steps described in previous sections, we obtain a set of flows. Using information about user actions, recorded during scripts execution (we describe it in Section 5.2.2), we are able to label flows after the user action that generates them. To do that we introduce the concept of *time interval* of a user action. A time interval starts in the instant when a specific user action is executed, and it ends after a timeout *TAA* (Time After a user Action). We consider all the flows within a time interval as related to that user action. So we label those flows after that user action. We also consider the flows within the time interval between *TAA* expiration and the instant of the next user action execution. These flows are not related to a user action, but we do not discard them. Indeed, we label these flows as if they belong to a "bogus" user action, we named *Background traffic*.

Time interval idea is also reported in the Figure 4.2, where with t we mean the time waited before the execution of the next user action. More details about the value for the timeout *TAA* can be found in Section 5.3.3.

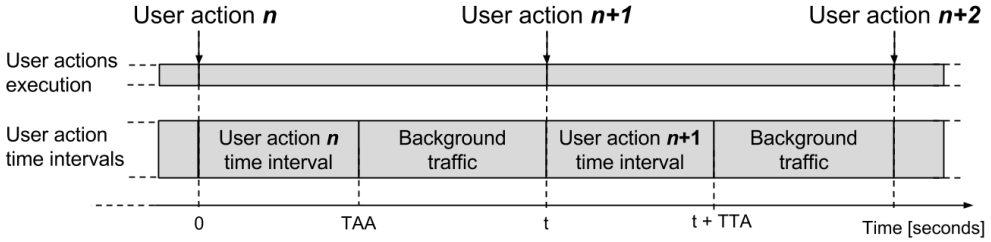

FIGURE 4.2: Time intervals related to user actions.

Finally, we create a file for every user action label, where we save flows marked with the same label.

4.3 Classification of user actions

Since we use a supervised learning approach, it is necessary to create a labeled dataset that describes the user actions that we want to classify. For each app that we analyze, we focus on user actions that are significant for that particular app.

4.3.1 Flows clustering

In most cases, a single user action generates a set of different flows (i.e., not just a single one). Furthermore, different user actions may generate different sets of flows. Our classification method is based on the detection of the sets of flows that are distinctive of a particular user action. In order to elicit these distinctive sets of flows, we build clusters of flows using the hierarchical clustering approach described in Section 3.2.

In the following, we describe the procedure we follow to build clusters of similar flows, also reported with the flowchart in Figure 4.3. Using clustering, flows that

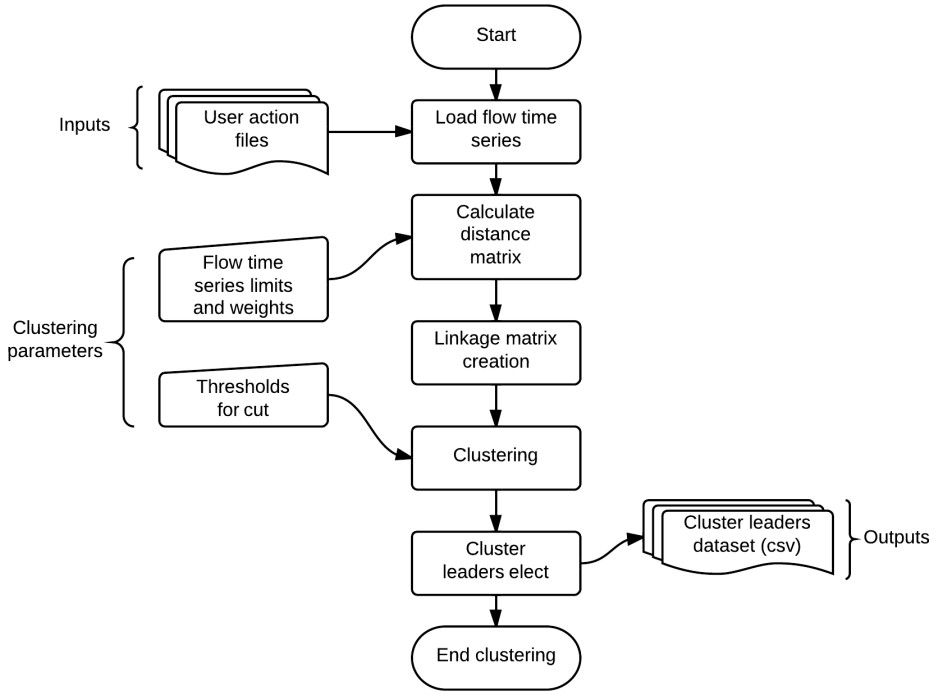

FIGURE 4.3: Flowchart for flow clustering procedure.

are similar one to each other will be grouped together, while not similar flows will be divided in different clusters. The average distance is used as linkage criterion,

while the computation of the distance between two flows combines the distances of the corresponding time series. Supposing that each flow f_i is decomposed into a set of n time series $\{T_1^i, \ldots, T_n^i\}$, the distance between f_i and f_j is defined as:

$$dist(f_i, f_j) = \sum_{k=1}^{n} w_k * DTW(T_k^i, T_k^j),$$

where w_k is a weight assigned to the particular time series. Weights can be assigned in such a way to give more importance to some type of time series with respect to others. For example, it is possible to give more weight to the time series that represent incoming packets, and less weight to those that represent outgoing packets.

We use thresholds, in terms of flows distance, to "cut" the hierarchy of clusters. So we obtain a set of clusters (also called flat clusters) for each considered threshold. The smaller the value of a threshold, greater will be the number of clusters in that set. In order to reduce the computational burden of the subsequent classification, a leader is elected for each cluster. A leader is a flow that represent its cluster. Given a cluster C containing the flows $\{f_1, \ldots, f_n\}$, we elect the leader selecting the flow f_i that has minimum overall distance from the other members of the cluster, that is:

$$\arg\min_{f_i \in C} \left(\sum_{j=1}^{n} dist(f_i, f_j) \right).$$

4.3.2 User actions classification

Clustering is executed over the set of flows used to build the training dataset. In the other hand, the cluster leaders are used to build both the training and the test datasets. We consider the user actions as instances of the datasets, while the class of each instance is a label representing the user action. We use one integer feature for each cluster identified through the clustering procedure. The value of each feature is determined analyzing the flows related to a user action. Each flow f captured after the execution of a user action will be assigned to the cluster that minimizes the distance between f and the leader of the cluster. The k^{th} feature indicate the number of flows assigned to the cluster C_k. For example, for the user action *send mail*, the k^{th} feature will be equal to 2 if there are 2 flows labeled with *send mail* assigned to the cluster C_k.

Finally, we execute the classification with naive Bayes and Random forest algorithms. The main idea behind the overall approach is that different user actions

will "trigger" different sets of clusters. The classification algorithms therefore learn which are these sets, and will be able to correctly determine the class labels for unseen instances.

Chapter 5

Data acquisition and analysis

In the first part of this chapter, we discuss the environment configuration and the procedure used to acquire network traffic. Later, we present the methodology followed to simulate user actions using scripts. Finally, we report some study about the domain of network traffic and user actions, in order to extract knowledge to improve performance of our framework.

5.1 Network traffic acquisition

In this section, we describe the environment we set up for data acquisition, from both hardware and software points of view. Starting with the description of hardware components, we conclude this section talking about apps and accounts.

5.1.1 Hardware and network configuration

For the evaluation of our solution, we use a Galaxy Nexus (GT-I9250) smartphone, running the Android 4.1.2 (Jelly bean) operative system, where we enable the "*Android Debug*" option. Being a "Google" phone, this device runs an original version of Android OS, without any other proprietary software. Anyway, we recall that in our approach we discard flows having an IP address with domain not related to a considered app (as explained in Section 4.1.1). We use a Wi-Fi access point (U.S. Robotics USR808054) to provide wireless connectivity to the mobile phone. Finally, we use a server (Intel Pentium Processor dual core E5400 2.7GHz with 4 GB DDR2 RAM) with two network cards running Ubuntu Server 11.04 LTS to route the traffic

from the access point to the Internet, and vice versa. To eavesdrop network packets flowing through the server, we use Wireshark software. From a Wireshark capture file, we create a comma separated file (csv), where each row describes a packet captured from the access point's interface. For every packet, we report source and destination IP addresses, ports, size in bytes and time in seconds from Unix epoch[1], protocol type and TCP/IP flags.

5.1.2 Account and apps setup

For our study, we consider three popular apps installed from the official Android market: Gmail v4.7.2, Facebook v3.8, and Twitter v4.1.10. For each app, we create 10 accounts that are divided in two different categories of users: "active" and "passive" users. An "active" user simulate the behavior of a user that actively use the app sending posts, email, tweets, surfing the various menus, etc. Differently, a "passive" user simulate the behavior of a user that passively use the app, just receiving messages or posts. The accounts of both passive and active users are configured in such a way to have several friends/followers within the group. We do not configure the accounts with actual friends or followers, in order to avoid interference due to notifications of external users activities that are not under our control.

5.2 Simulation of user actions using scripts

In this section, we describe the methodology we follow to simulate user actions on Android apps. For the comprehension of this section, it is necessary to clarify the distinction we make between interaction and user actions: with the term "interaction" we refer to a physical or simulated operation on a device (e.g., tap, swipe, key press), while with "user action" we mean an operation done by user on an app (e.g., send an email, post a status), that could require several interactions to be achieve.

We simulate interactions on a device issuing commands from Android Debug Bridge (ADB). Provided by Android SDK, ADB is a versatile command line tool used to control an Android device. This tool uses a client-server paradigm with these components:

- client runs on a machine, and it can be invoked from a shell by issuing an ADB command;

[1] 00:00:00 UTC, January 1st, 1970

- server runs as a background process on a machine, this process manages communication between ADB daemon on device and the client;
- ADB daemon runs on device, and execute commands issued by the client.

In our experiments, we first considered *MonkeyRunner* tool to simulate interactions on a device. Unfortunately, we observed a conflict between *MonkeyRunner* daemon and Gmail app, in fact some commands like sending an email (tap on "send" button) are not executed properly. For this reason we abandoned *Monkeyrunner* in favor of ADB tools.

Every user action could be consider as a sequence of interactions with a device. Indeed, sometimes it is necessary a sequences of interactions in a precise order to achieve a specific purpose, but often it could be achieve with a single interaction. For example, to compose a textual content in a message it is necessary to issue a sequence of "key press" events (one for each characters in the text). In the other hand, to open in-box message page it is sufficient a single tap on a set of specific coordinates on the screen.

Our scripts are composed by a sequence of four basic ADB commands: tap, swipe, key press and process kill. In Listing 5.1, we report the Python functions to invoke these commands.

```
from subprocess import call

# simulation of a tap on coordinates (x, y) on screen
def adb_touch_tap(x, y):
    cmd="adb shell input touchscreen tap %s %s" % (x, y)
    call(cmd.split())
    return

# simulation of a swipe from coordinates (x1, y1) to (x2, y2) on screen
def adb_touch_swipe(x1, y1, x2, y2, speed):
    param=(x1, y1, x2, y2, speed)
    cmd="adb shell input touchscreen swipe %s %s %s %s %s" % (param)
    call(cmd.split())
    return

# simulation of keyboard button press identified by "code"
def adb_keycode(code):
    cmd="adb shell input keyevent KEYCODE_%s" % (code)
    call(cmd.split())
    return

# sending a signal to kill a process
def kill(process):
    cmd="adb shell am force-stop %s" % (process)
```

```
    call(cmd.split())
    return
```

LISTING 5.1: Python fuctions to execute ADB shell commands.

For each app, we created a specific script, where we define a sequence of user actions. During the script execution, these user actions are performed like there is an human user doing them.

When a script performs a sequence of user actions, the app pass through different states. We call *states flow* the whole sequence of these states observed during the execution of a script. From start of an app, a script induce the app in a specific states flow. We do that to be sure to predict the positions of all components in the layout of the app when it is on a precise status. For example, given an app state and the next interaction that is a tap on a button object A, the coordinates of object A must remain the same during all script executions.

We use *Android Device Monitor* tool to obtain the precise coordinates of every objects in a layout of an specific app status. Then, we collect these coordinate in a database that is available for all scripts for an app. Unfortunately, this procedure is slow since it does not exists an automatic tool to do what we need. Two examples for layout objects coordinates obtaining procedure are reported in Figures 5.1 and Figure 5.2, which represent screen coordinates for "Important" object in menu layout on Gmail app, and a tweet in "Discover" page on Twitter app respectively.

5.2.1 User actions sequence

To reach a particular target, a user may have to perform several user actions in a precise order. For example, a user has to perform three user actions in a precise sequence to post a message on his Facebook wall (also reported on Figure 5.3). He has to be sure that the Facebook app shows the "user's wall", then he has to tap on the "write a post" button (1), fill the edit box with some text (2), and finally tap on the "Post" button (3).

All user actions in a sequence are scripted specifically for each app and its layouts, maintaining a consistent and predicable states flow. The scripts explore several functionality of each app.

It is important to highlight that we do not use static text as content of emails, posts, tweets or direct messages. indeed, we use textual strings randomly selected

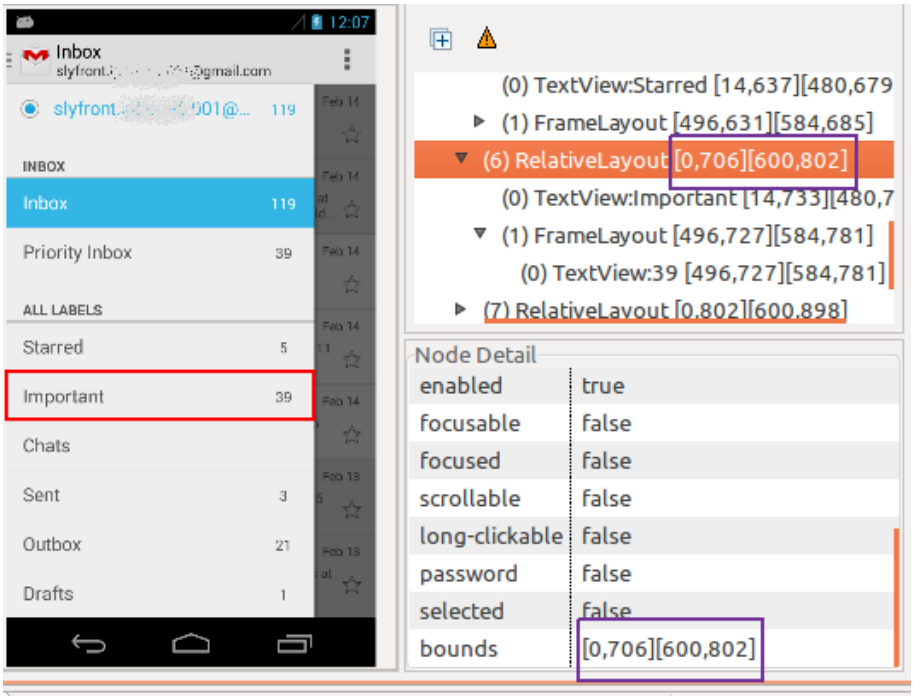

FIGURE 5.1: Coordinates for an object in Gmail layout.

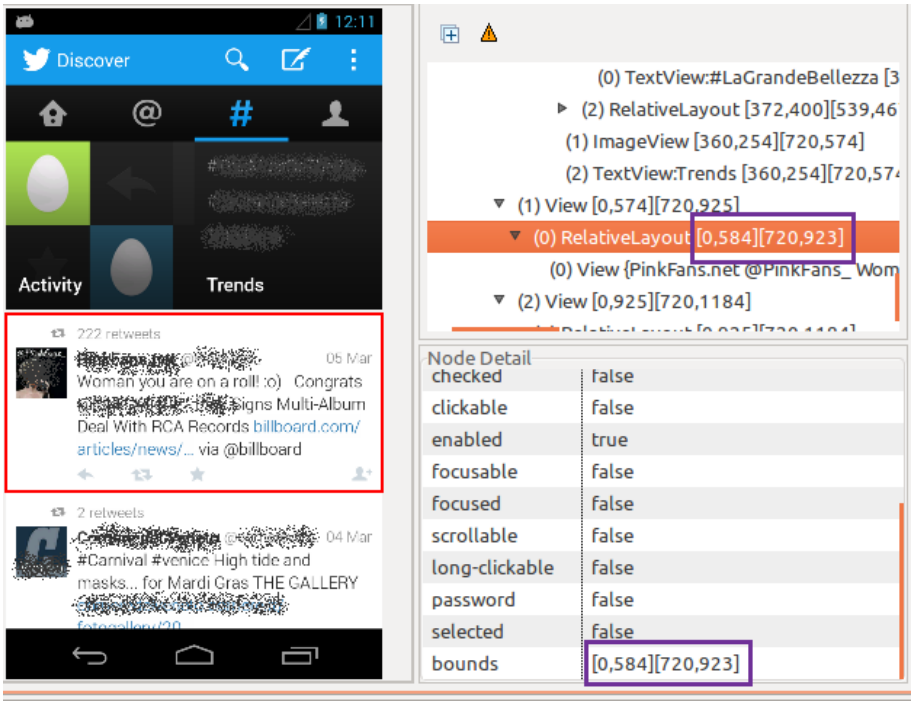

FIGURE 5.2: Coordinates for an object in Twitter layout.

from a large set of sentences (limited to 140 characters on Twitter app), but we do not consider the images upload, neither of files.

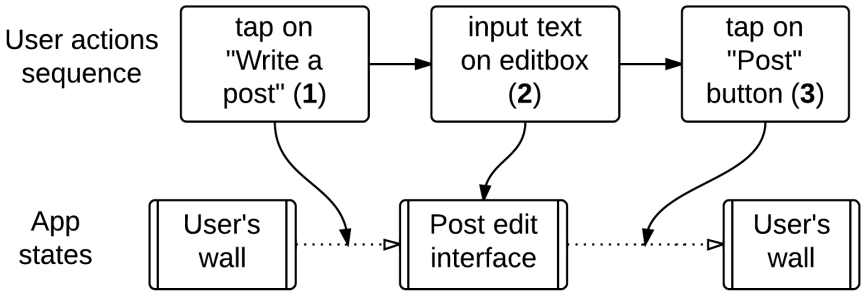

FIGURE 5.3: States flow on Facebook app due to user actions.

5.2.2 Scripts execution

Every script iterate at least 20 times a whole sequence of user actions for an app. During scripts execution are produced log files, where a row corresponds to a user action. In these log files, script record for each row the label of the executed user action, the sequence id (script iteration), the time stamps from Unix epoch and from start of the sequence. Examples of log files produced by a script are reported in Appendix A.

Between user actions in a sequence there is a wait time t equal to 20 seconds, this permit to catch all the traffic generated by a user action, avoiding any interference due to the execution of the next one. It is very important to maintain the synchronization between time stamps on log files and Wireshark captures. To do that we execute both scripts and Wireshark capture on the same machine, where system time is periodically updated using a ntp server.

5.3 Domain study

In this part of the chapter, we describe the studies done to understand the domain of our analysis. After data acquisition process, we analyze collected data to improve flows clustering and user actions classification. To do so, we start studying the features of small entities like packets, then traffic flows, and gradually expanding the focus until the user actions. We conclude observing user actions properties from the point of view of network traffic, in order to find which features can discriminate user actions from each other.

5.3.1 Packets protocols

In Chapter 2, we report some works that use payload analysis on HTTP packets, in order to trace a user browsing on a OSN [41, 42, 43]. A similar technique is also used to profile Android apps [51, 52]. Being this techniques not feasible on payload encrypted using SSL/TLS protocols, we analyze the protocols distribution over the packets. In this analysis, we consider packets after filtering procedure, where TCP connection control packets are discarded (as previously described in Section 4.1.2). We report in Table 5.1 the distribution of protocols over network packets captured.

Protocol	**Facebook**	**Gmail**	**Twitter**	**Total**
TLSv1	96.0% (33398)	90.9% (7205)	96.3% (13435)	95.4% (54038)
TCP	03.7% (1316)	00.8% (70)	03.7% (522)	03.4% (1908)
HTTP	00.0% (0)	08.1% (644)	00.0% (0)	01.1% (644)
SSLv2	00.1% (63)	00.0% (0)	00.0% (0)	00.1% (63)

TABLE 5.1: Protocol presence among packets captured for all considered apps.

From the analysis on packets protocols, we observe that more than 95% of packets are encrypted with TLS protocol, but 90% of packets related to Gmail app. In the other hand, only Gmail app produces packets using HTTP protocol. For this reason, an analysis based on packet payloads is not feasible on network traffic produced by the apps we consider.

5.3.2 Traffic flows

Flow are fundamental entities of our analysis, and they can be seen as a sequence of network packets ordered by time. As we explained in Section 4.1.3, we build three time series using the sizes in bytes of packets in a flow. The final purpose of the studies about flows is to understand how to regroup them in sets using their features. In the following, we present a possible representation for a flow and the notation we use to identify an interval of packets in a flow.

Representation of a flow

Being a flow a sequence of packets, we build an object with methods and data fields projected for statistical feature analysis, in anticipation of future studies. In

effect, we use *Series* objects from *pandas*[2] libraries, for data fields corresponding to time series like incoming, outgoing sequences of packets or their time-gap. *Series* class provides methods for statistical data analysis tools on these time series, which were useful on our preliminary studies of flows. We reports an example of time series generated by three flows in Table 5.2. Values within square brackets represent the amount of bytes exchanged per packet: negative values in complete time series indicate incoming bytes, while positive values indicate outgoing bytes.

Flow ID	**Time series type**	**Time series**
Flow 1	Incoming	[1514, 1514, 315, 113, 477]
	Outgoing	[282, 188, 514, 96, 1514, 179, 603, 98, 801, 98]
	Complete	[282, -1514, -1514, -315, 188, -113, 514, ..., 96, 1514, 179, 603, 98, 801, 98, -477]
Flow 2	Incoming	[1514, 1514, 1266, 582, 113, 661]
	Outgoing	[282, 188, 692, 423]
	Complete	[282, -1514, -1514, -1266, -582, 188, -113, ..., 692, 423, -661]
Flow 3	Incoming	[1245, 1514, 107, 465, 172, 111]
	Outgoing	[926, 655, 136, 913, 1514, 1514, 863]
	Complete	[926, 655, 136, -1245, 913, 1514, 1514, 863, ..., -1514, -107, -465, -172, -111]

TABLE 5.2: Example of time series generated by three flows.

In Figure 5.4, we graphically represent the same three flow time series through a cumulative chart. The lower side of the chart represents incoming traffic, while the upper side represents outgoing traffic. This is only one of the possible representations, and it shows that the "shapes" of these three flows are quite different. Intuitively, flow clustering procedure aims to identify the "shape" of unseen flows.

Flow packets interval

In a flow, *packets interval* is a representation we use to specifies the first and the last packet to be considered. For example, considering a flow f composed by l packets, and the interval $[x, y]$ with $x \leq y$ and $y \leq l$, the corresponding time series will be composed by $y - x + 1$ values that report the bytes of the x^{th} to the y^{th} packet. This simple mechanism allows us to focus on particular portions of the flow. The first part of a flow is often the more significant. Indeed, Lim et al. report that

[2] pandas is an open source library providing data structures and data analysis tools for the Python programming language – http://pandas.pydata.org/

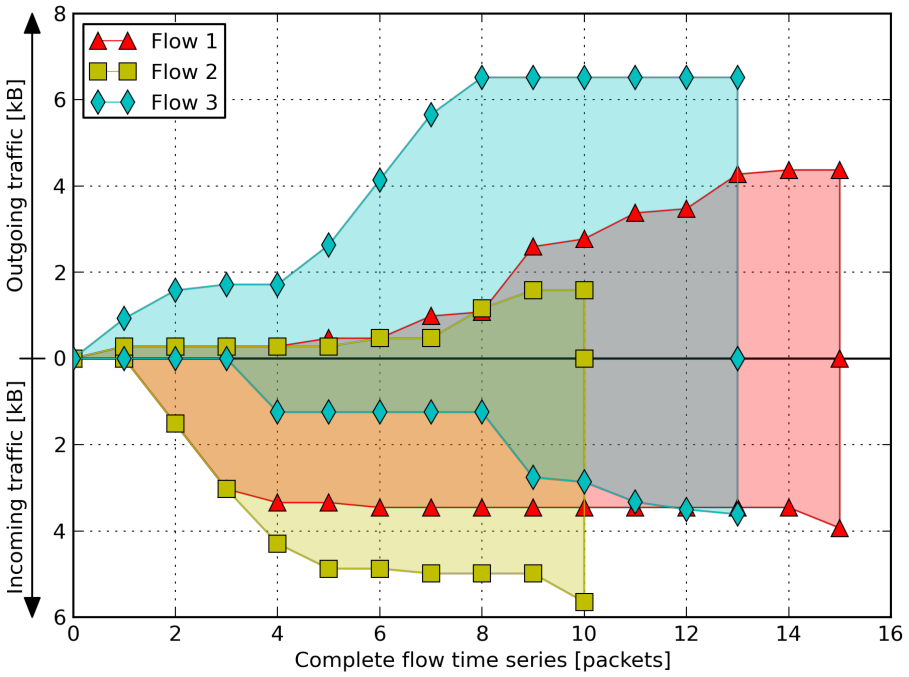

FIGURE 5.4: Representation of flows time series.

the first packets in a network flow are a significant feature in traffic analysis [26]. We use this representation to identify interval of packets considered for clustering parameters configuration (in Section 6.1.2), showing that the best configuration is app dependent.

5.3.3 User actions

In this section, we present two studies on flows related to a user action. First, we discuss about the duration of a time interval of a user action. Then, given different time intervals of the same user action, we focus on sets of similar flows within them.

User actions time intervals

In Section 5.2.1, we report that in a script, there is a wait time t equal to 20 seconds between the execution of a user actions and the next one. Later, during our studies on network traffic, we realize that an interval of 20 seconds after a user action execution are too many. Indeed, it is hard to believe that a user action could trigger the transmission of new flows after 20 seconds from its execution. It is more reasonable to consider the flows that starts after a timeout as background traffic, generated by the app during inactivity periods. In order to confirm this intuition,

we study flows distribution over time in time intervals related to user actions. In Figure 5.5, we report a chart where we overlap flows seen after the executions of 40 user actions per type. Every flow is represented through a cumulative chart, similar to the one used in Figure 5.4, in the domain of time.

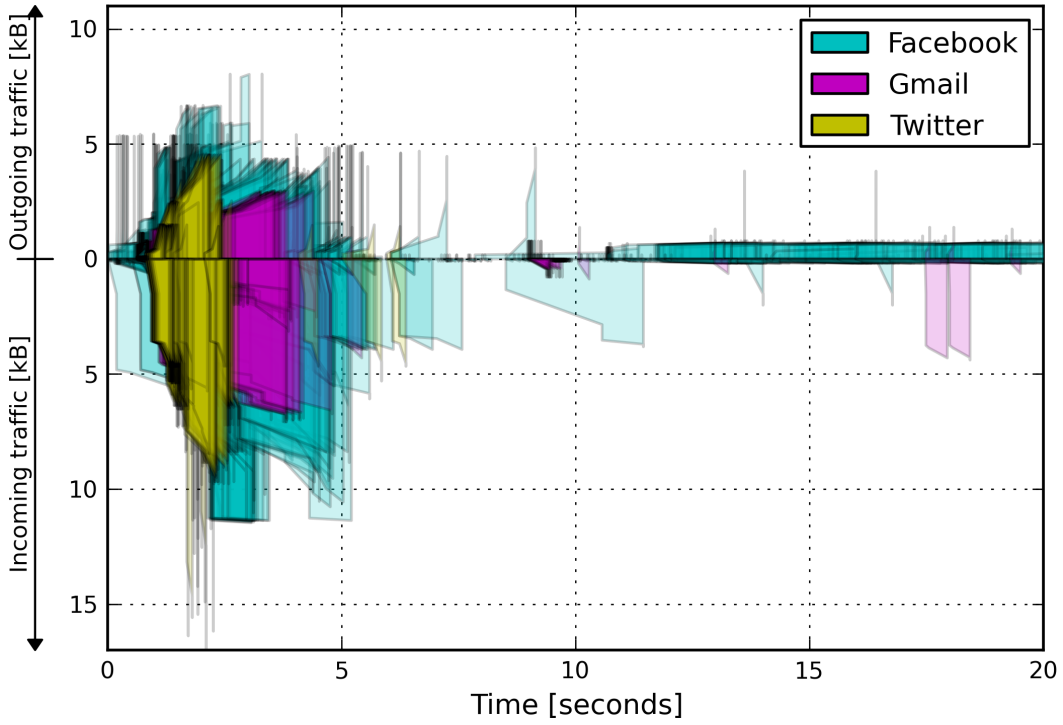

FIGURE 5.5: Network flows over time.

In this chart, it is possible to observe that network traffic concentrates on the first 6 seconds, with a lapse of "silence" after. To confirm that impression, we study the presence of flows in the domain of time, using the same data of the previous chart. We do a sampling every 0.2 seconds, counting the number of "active" flows in every sample of time. We report this alternative representation of data the chart in Figure 5.6. As expected, there is a concentration of flows after the few seconds after the execution of a user action. Afterwards the number of flows decreases reaching a minimum after 6 seconds (especially Facebook app). For this reason, we set the value of TAA timeout to 6 seconds, also highlighted with a red dashed line in the chart in Figure 5.6.

So we consider the flows that start within the time interval ranging from 0 to 6 seconds, as directly generated by the execution of a user action.

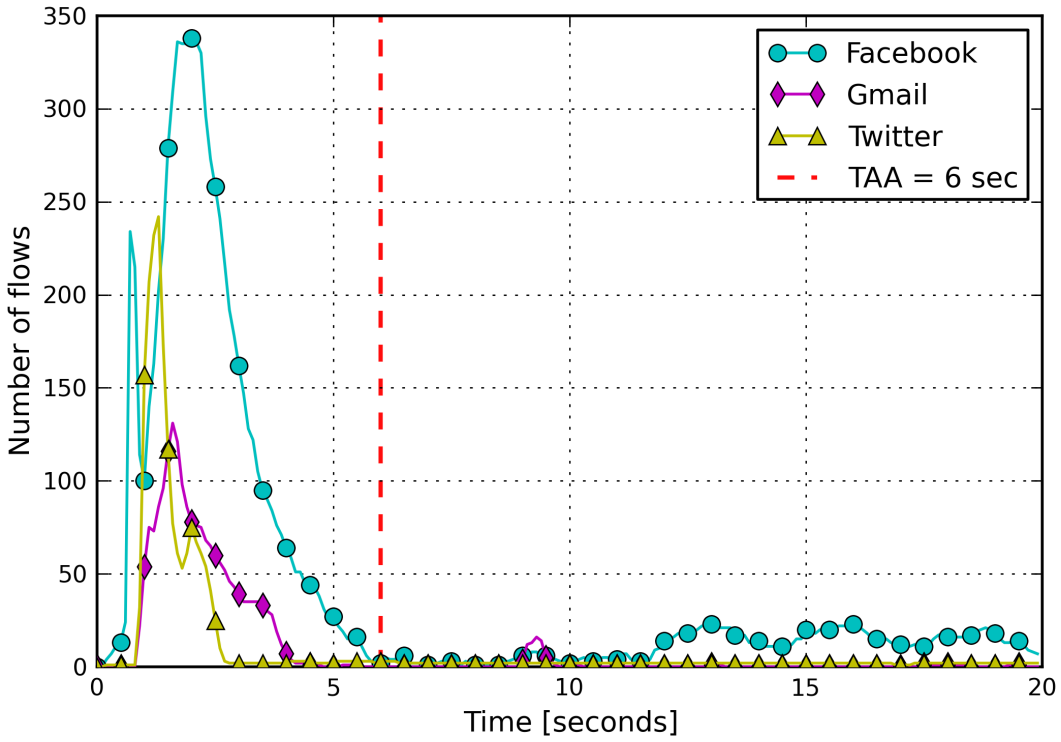

FIGURE 5.6: Presence of flows over time.

Flows clustering on user actions

Analyzing time intervals of single user actions, we often found multiple flows within them. An example for this assertion can be found on *post on wall* user action on Facebook app. In Figure 5.7, we represent in a chart the flows within the time interval of a single "post on wall" user action, where the complete time series of the those flows are:
Flow 1 → [92, 92, 93, 108, 93, 135, -95, 92, 92, 93, 108, 93, 134, -95];
Flow 2 → [928, 609, 137, -1337, 746, 353, -143, -1514, -1192, 764, -874, 822, -590].

Observing the different "shapes" for the flows, we understand that it is possible to distinguish flows in categories using time series. We show that using hierarchical clustering on flows generated by 20 user actions with the same label, in this case *post on wall*. In this study, we consider complete flow time series and DTW distance as metric (more precisely the cost of *optimal warping path*), while we use average as linkage criterion.

The dendrogram in Figure 5.8 illustrate the arrangement of the clusters produced by hierarchical clustering. From the bottom to the top of the dendrogram, it is possible to observe how single flows are aggregate in clusters. A node indicates

a link between two clusters, merged in a new one. The height of a node coincide with the average DTW distance between the complete flow time series inside the new cluster.

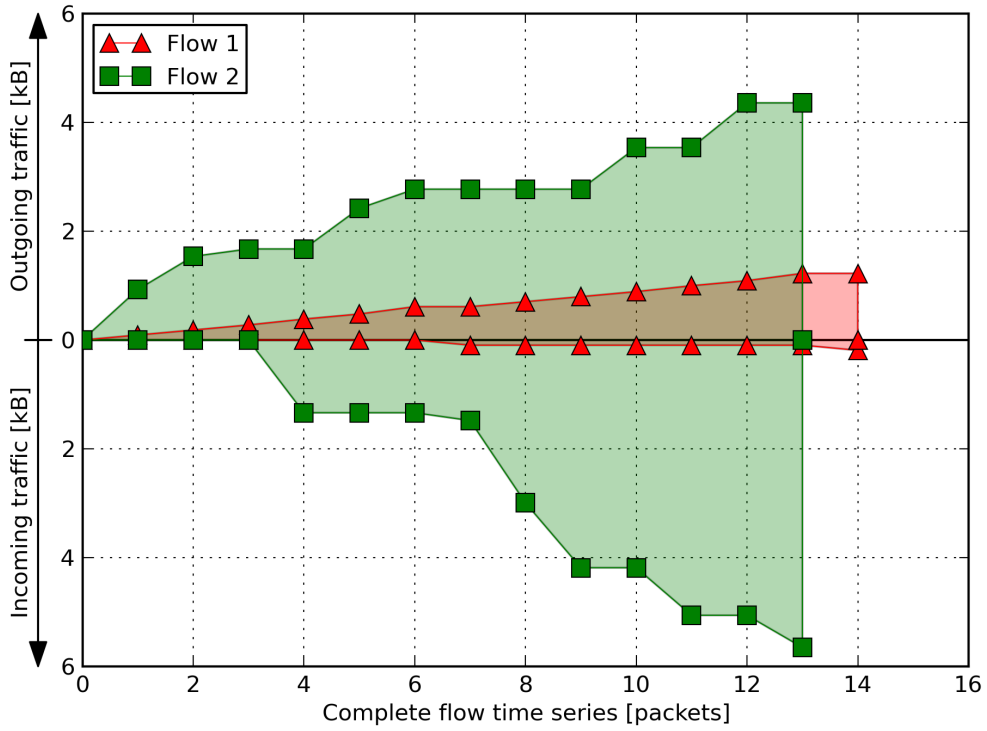

FIGURE 5.7: Flows related to an *post on wall* user action on Facebook.

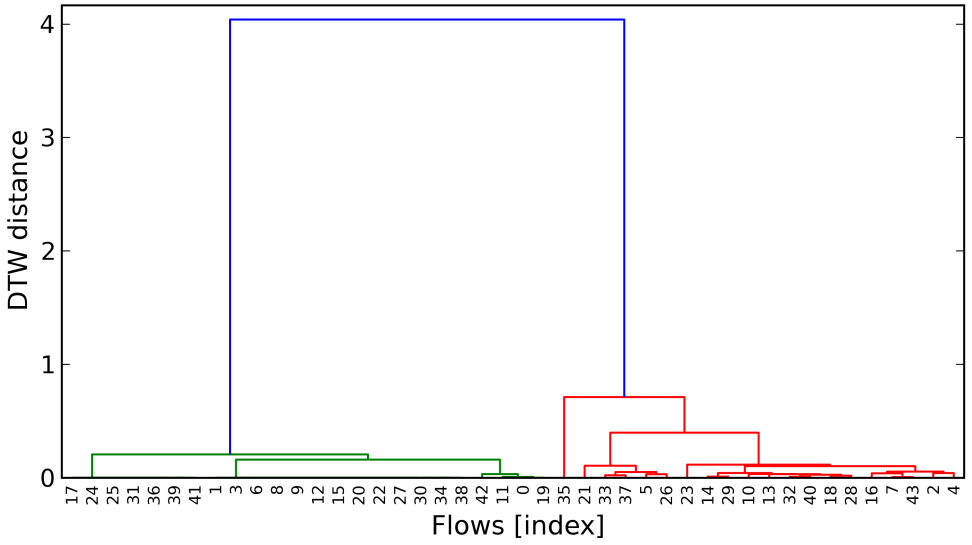

FIGURE 5.8: Dendrogram of hierarchical clustering of flows for *post on wall* user action on Facebook.

As we expected, we can observe on dendrogram two different sets of similar flows (marked with different colors). So we verified that it is possible to elicit similar flows using DTW distance between flow time series. In this example, we apply clustering algorithm on flows related to a single user action. In our framework, we consider instead all user actions at the same time, calculating distance between flows with the method reported on Section 4.3.1).

Chapter 6

Clustering and classification

In this chapter, we describe the implementation details about flows clustering and user actions classification. First, we present flows clustering parameters selection and implementation choices. Afterwards, we discuss about the set up and performance comparison between algorithms we use for user action classification. Finally, we report the tool and libraries we use in the implementation of the clustering and classification procedures.

6.1 Unsupervised clustering

In the overview of our framework in Chapter 4, we introduce the flow clustering procedure. To regroup flows in sets, it is necessary to use an unsupervised clustering, because we do not have any knowledge about the these sets, nor their number. First of all, we have to introduce a study on flows length distribution, in order to explain which part of time series we have to consider. We proceed defining the clustering parameters to calculate the distance between flows.

6.1.1 Statistical flow length distribution

A further study on flows time series relates to their length. Figures 6.1, 6.3 and 6.2 reports the statistical distribution of the length of the complete, outgoing and incoming flows time series respectively. In these plots, the first and third quartile are represented as the left and right side of the notched box. The notch of the box represents the median value. Lines that extend horizontally from the boxes indicate

the 2nd percentile (left) and the 98th percentile (right). This knowledge is useful to identify which part of a flow time series could be more significant.

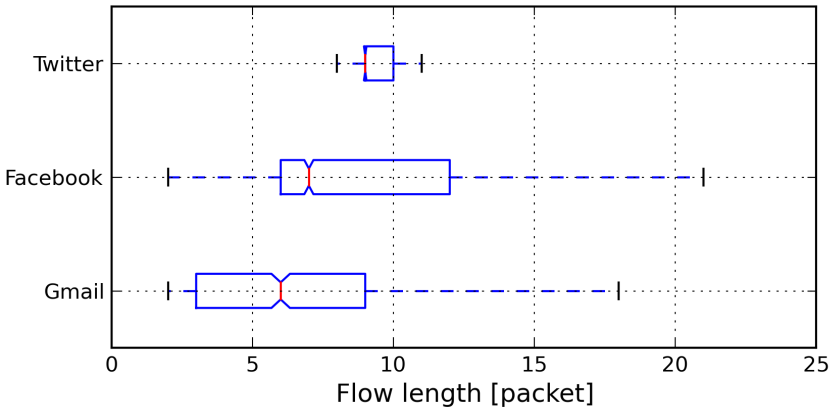

FIGURE 6.1: Statistical distribution of the length of complete flow time series.

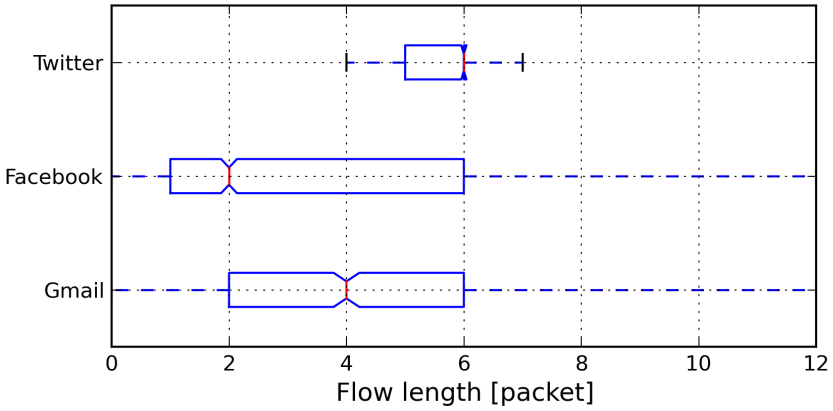

FIGURE 6.2: Statistical distribution of the length of incoming flow time series.

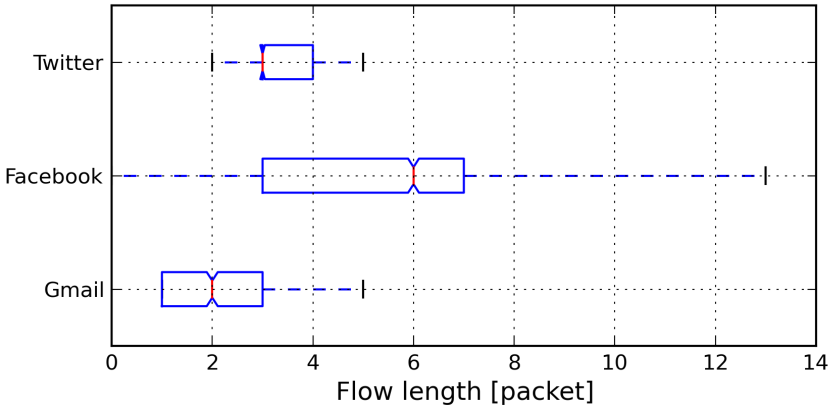

FIGURE 6.3: Statistical distribution of the length of outgoing flow time series.

6.1.2 Clustering parameters selection

Using knowledge obtained from the study of network flow features, we select a set of parameters to measure the distance between these flows. As explained in Section 4.1, each network flow is modeled as a set of time series: incoming, outgoing and complete. Table 6.1 reports the weights and the packet intervals considered of each app. The notation used for specify the interval of packets to consider is the same we introduced in Section 5.3.2.

Apps	**Sets**	**Weights**	**Incoming**	**Outgoing**	**Complete**
Gmail	Conf. 1	0.80 0.20	[1,4] [1,6]	[1,2] [1,3]	[1,6] [1,9]
	Conf. 2	0.66 0.33	[1,4] [1,6]	[1,2] [1,3]	[1,6] [1,9]
	Conf. 3	0.33 0.66	[1,4] [1,6]	[1,2] [1,3]	[1,6] [1,9]
Facebook	Conf. 1	0.66 0.33	[1,3] [1,6]	[1,5] [1,7]	[1,7] [1,12]
	Conf. 2	0.33 0.66	[1,3] [1,6]	[1,5] [1,7]	[1,7] [1,12]
	Conf. 3	0.20 0.80	[1,3] [1,6]	[1,5] [1,7]	[1,7] [1,12]
Twitter	Conf. 1	0.95 0.05	- -	- -	[7,10] [1,10]
	Conf. 2	0.95 0.05	- -	- -	[8,11] [1,11]
	Conf. 3	0.95 0.05	- -	- -	[8,10] [1,10]

TABLE 6.1: Weights set configurations and packets intervals for Gmail, Facebook and Twitter apps.

We use different weights configurations, and we select the maximum number of packets to consider in each time series analyzing the statistical length of the flows. We consider the median value and the third quartile as thresholds to limit the length of the generated time series.

Analyzing Figure 6.1, it can also be noticed that the the length of the flows generated by Twitter is always between 9 and 10 TCP/IP packets. In fact, this app seems to generate the same type of flows independently of the particular user action that is executed. To confirm this statement, we report in Figure 6.4 and Figure 6.5 the graphical representation of the flows that occur when executing three different user actions on Gmail and Twitter apps respectively.

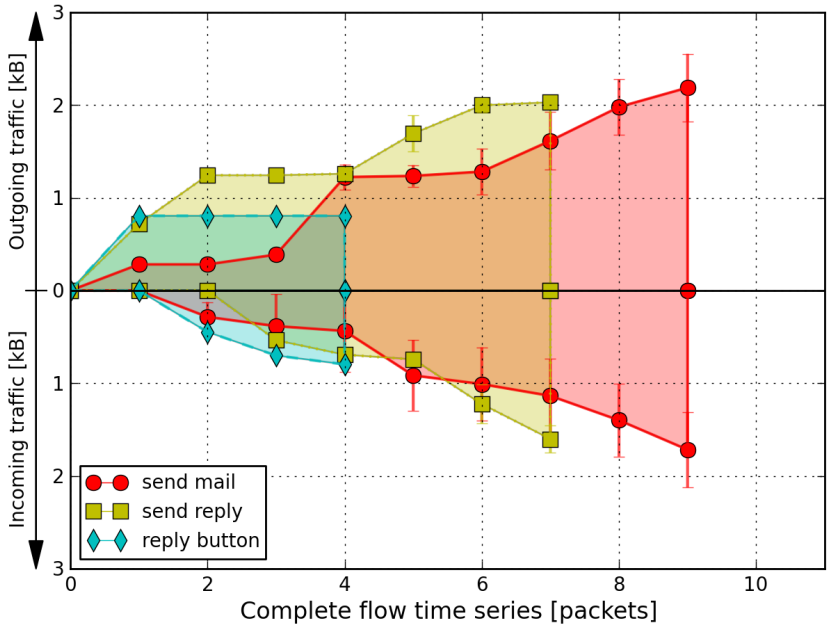

FIGURE 6.4: Representation of three different Gmail user actions.

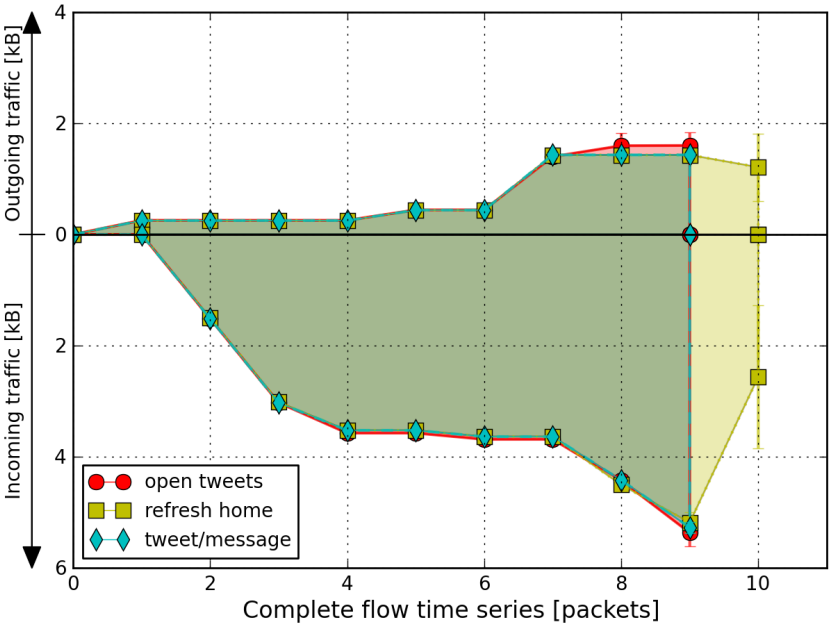

FIGURE 6.5: Representation of three different Twitter user actions.

Comparing the two figures, we notice that the shape of flows produced by the user actions drastically change on Gmail app, while they are almost unvaried on Twitter app. As a matter of fact, different Twitter user actions just differ in their last three or four packets. Nevertheless, our approach reaches very good performance also for this app.

6.2 Supervised classification

In this section, we present implementation details about the classification of user actions. First, we define the division of the collected data in training and test sets. We also report the selection of the significant user actions for every app. Finally, we present the results of the challenge between naive Bayes and Random forest, in terms of classification accuracy.

6.2.1 Training and test sets

During our experiments, we collected and labeled the traffic generated by 220 different sequences of user actions for each app (for a total of 11660 different user actions for Gmail, 6600 for Twitter, and 10120 for Facebook). The 70% of these user actions are used for the training set, while the remaining 30% are used for the test set.

In the test set we consider the traffic generated by accounts that are not used to generate the training set. Using different accounts to generate the training and the test set, it is possible to assure that the results of the classification do not depend on the specific accounts that have been analyzed.

6.2.2 Significant user actions

For each app, we choose a set of user actions that are in our opinion more sensitive than others. The list of user actions is reported in Table 6.2. Less important user actions are instead labeled with the term *other*. In the class *other*, there is also the flows labeled as *Background traffic*.

6.3 Classification algorithms comparison

In this section, we present the results of performance for the considered classifiers, in terms of accuracy. To understand which is the classifier with best accuracy between naive Bayes and Random forest, we test them on a validation set (disjoint from training and test sets). We report in Figure 6.6 and Figure 6.7 the averaged F-measure for every app, obtained from naive Bayes and Random forest classifiers respectively, when varying the number of clusters.

Facebook

User action	Description
send message	send a direct message to a friend
post user status	post a status on user's wall
open user profile	select user profile page from menu
open message	select a conversation on messages page
status button	select "*write a post*" on user's wall
post on wall	post a content on a friend's wall
open facebook	Facebook app execution start

Gmail

User action	Description
send mail	send a new email
reply button	button to reply selection
open chats	select chats page from menu
send reply	send a reply to an email

Twitter

User action	Description
refresh home	request for refresh the home page
open contact	select contacts page on menu
tweet/message	publish a tweet or send a message
open messages	select direct messages page
open twitter	Twitter app execution start
open tweets	select tweets page on menu

TABLE 6.2: Description of the Significant user actions for each app.

Considering the same number of clusters, we observe that for Facebook and Twitter apps the accuracy obtained from Random forest is higher than naive Bayes. While for Gmail app, the accuracy is quite the same for both classifiers.

Number of clusters

One of the issues to discuss before proceeding to the classification of the user actions is the number of clusters to consider. In order to establish a reasonable value for this parameter we study the Random forest performance (Figure 6.7). For each app, we therefore considered the number of clusters that maximized the accuracy, in terms of averaged F-measure.

6.4 Implementation tools

We implement flows clustering and classification procedure using external libraries for Python. To calculate DTW distance between a couple of time series, we use the

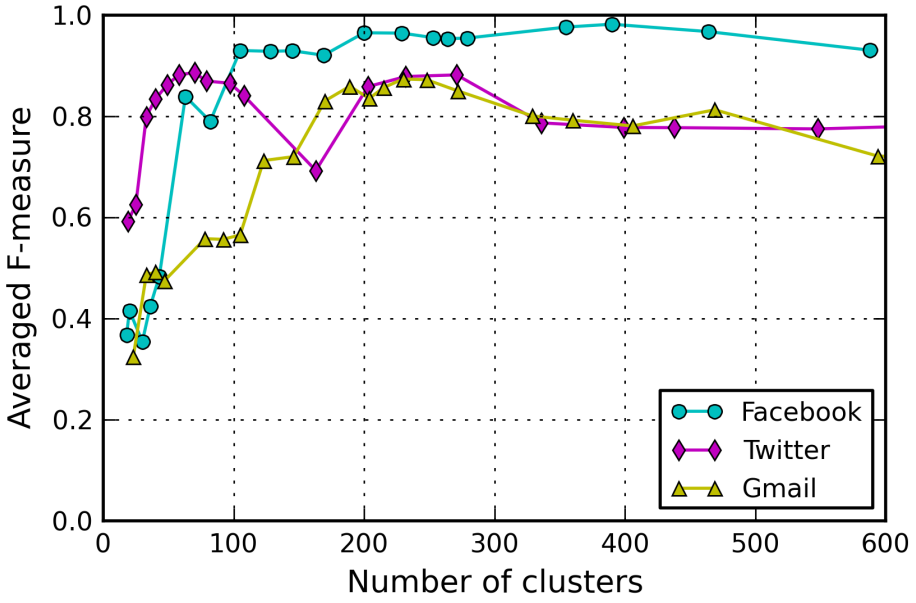

FIGURE 6.6: Classification accuracy using naive Bayes over number of clusters.

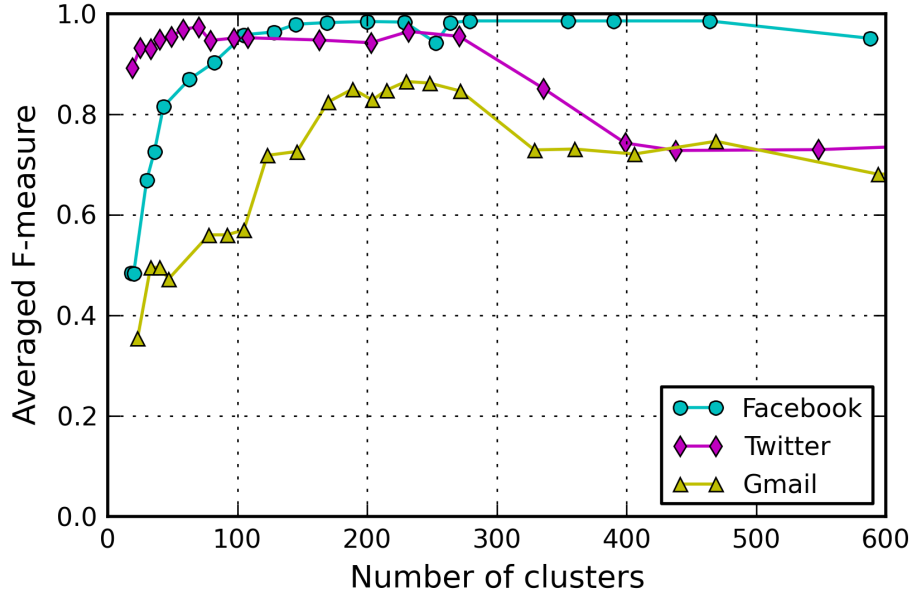

FIGURE 6.7: Classification accuracy using Random forest over number of clusters.

dtw_std function defined in *mlpy*[1] libraries v3.5.0.

Lots of Python libraries provide implementation for hierarchical clustering algorithms, but the best for our needs is the module *scipy.cluster.hierarchy* in *SciPy*[2]

[1]mlpy is a Python module that provides machine learning methods for supervised and unsupervised – http://mlpy.sourceforge.net/

[2]SciPy is a Python-based ecosystem of open-source software for mathematics, science, and engineering – http://www.scipy.org/

libraries v0.13.0.

In classification procedure, we exploit the classification algorithms implemented by *scikit-learn*[3] libraries v0.14. In particular, we use *GaussianNB* for naive Bayes and *RandomForestClassifier* for Random forest.

[3]scikit-learn are tools for data mining and data analysis – http://scikit-learn.org/

Chapter 7

Classification performance

In this chapter, we report the results of classification of user actions performed using Random forest algorithm. For every considered app, we discuss the performance the three considered configurations of weights and packets intervals, identifying the one with the best accuracy. For that configuration, we report in details results in terms of precision, recall and F-measure metrics. We describe these results using a table, where each row represent a user action for the app in object. Finally, we report the confusion matrix for that particular configuration. In the following sections, we report performance for Facebook, Gmail and Twitter apps, in this order.

7.1 Facebook

We focused on seven different user actions that may be sensitive when using the Facebook app. On average, the F-measure is equal to 99%, with a precision and a recall of 99% and 98% respectively. Performance reached with different configurations of weights and packets intervals constraints are reported in Figure 7.1.

For each user action at least one of the configurations exceeds 94% of accuracy, while the worst performing is always higher than 74%. Table 7.1 reports precision, recall and F-measure reached using Configuration 3.

We noticed that all the user actions have a precision higher 96%. The recall is higher than 95% for all the actions but the *open user profile*, that reaches 91%. In effect we realized that this particular user action is classified as *other* in 9% of the examples, as we can see from the confusion matrix reported in Figure 7.2.

User actions	Precision	Recall	F-measure
send message	1.00	1.00	1.00
post user status	1.00	0.95	0.97
open user profile	0.96	0.91	0.94
open message	0.98	1.00	0.99
status button	1.00	1.00	1.00
post on wall	1.00	0.98	0.99
open facebook	1.00	1.00	1.00
other	0.99	1.00	0.99
Average	0.99	0.98	0.99

TABLE 7.1: Classification results of Facebook user actions reached using Configuration 3.

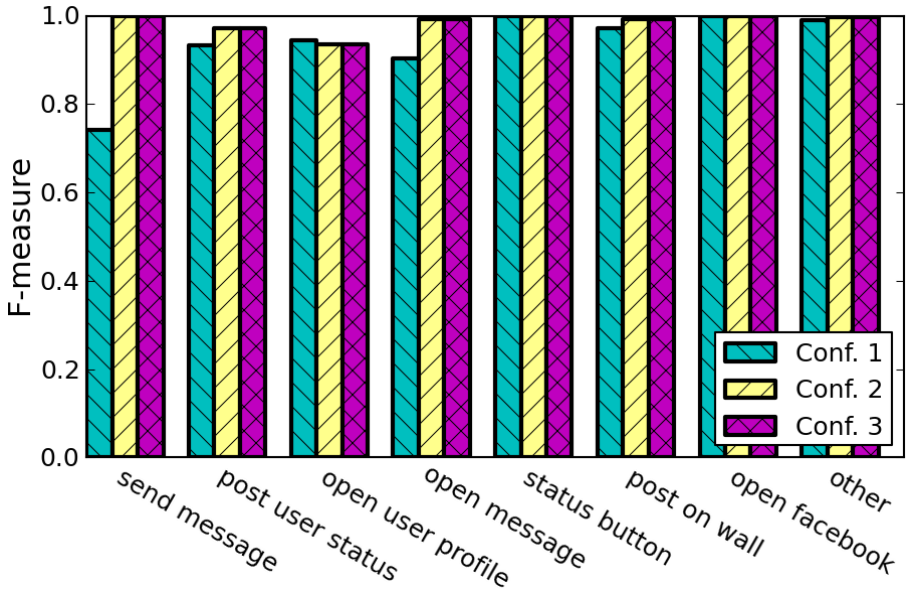

FIGURE 7.1: Classification accuracy of the Facebook user actions.

7.2 Gmail

In this section, we analyze four specific user actions of the Gmail app: *send mail*, *reply button*, *open chats* and *send reply*. Figure 7.3 shows the classification accuracy that has been reached. We observe that we are able to distinguish with high accuracy the user action of sending of a new email, from that of replying to a previously received message, as well as the tap over the reply button. The *open chats* user action is instead more difficult to distinguish.

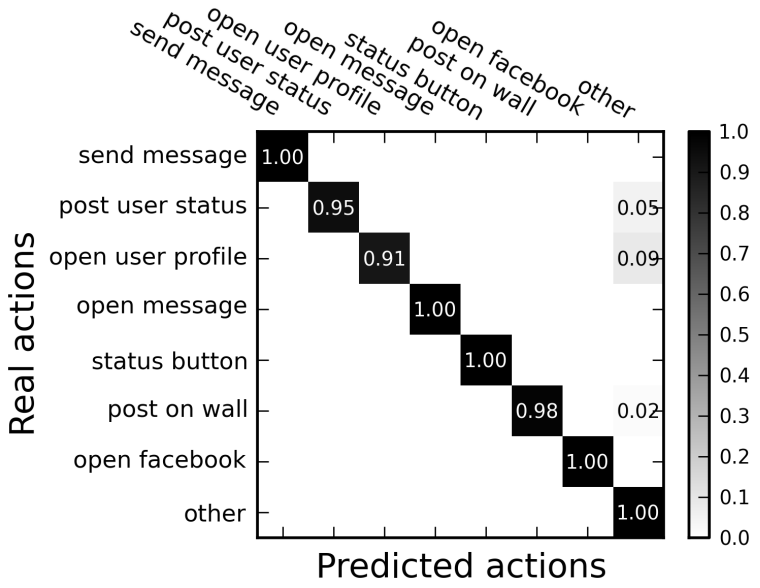

FIGURE 7.2: Facebook user actions confusion matrix for Configuration 3.

Table 7.2 reports precision, recall and F-measure for different configurations of weights and packets intervals constraints. We can observe that the user action *open chats* (that allows to read past chats) achieves a low precision but a high recall.

User actions	**Precision**	**Recall**	**F-measure**
send mail	1.00	1.00	1.00
reply button	0.85	1.00	0.92
open chats	0.36	0.94	0.52
send reply	0.98	1.00	0.99
other	0.99	0.82	0.90
Average	0.83	0.85	0.86

TABLE 7.2: Classification results of Gmail user actions reached using Configuration 1.

Analyzing the confusion matrix depicted in Figure 7.4 it is possible to notice that 16% of *other* user actions wrongly classified as *open chats*. This is the reason of such a low precision.

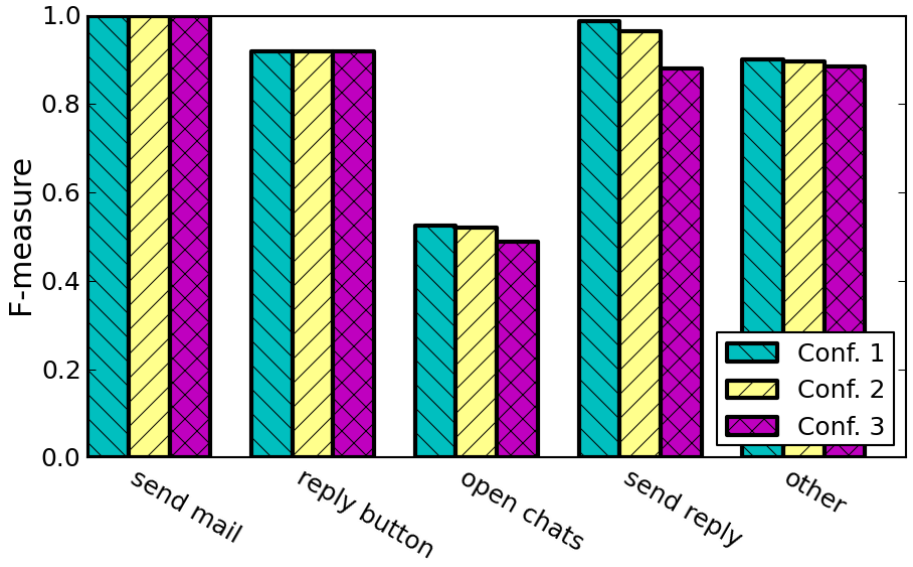

FIGURE 7.3: Classification accuracy of the Gmail user actions.

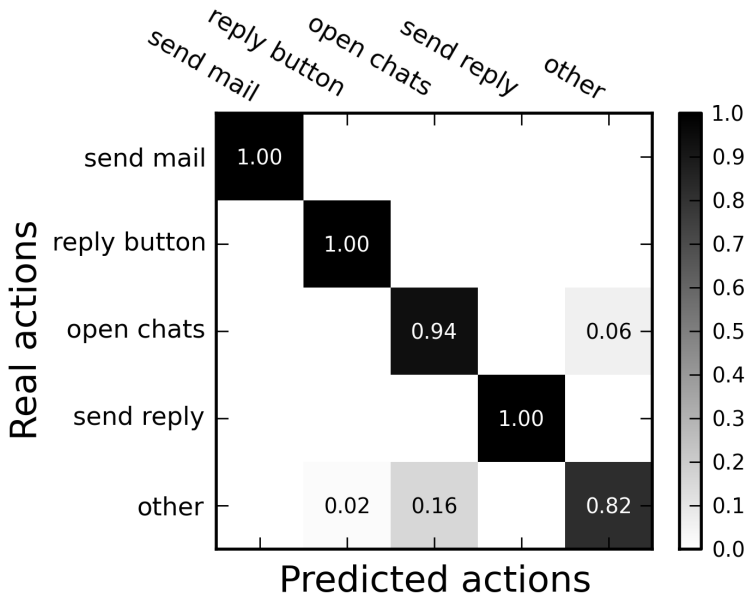

FIGURE 7.4: Gmail user actions confusion matrix for Configuration 1.

7.3 Twitter

During the analysis we noticed that Twitter user actions may be more difficult to classify than Gmail and Facebook user actions. Indeed, different Twitter user actions generate similar time series that have in common a large portion. Only the last three or four packets of each time series show some difference. Nevertheless, we have been

able to reach outstanding results also for this app. In particular, we focus on six specific user actions: *refresh home*, *open contacts*, *tweet/message*, *open messages*, *open twitter*, *open tweets*.

On average, the F-measure is equal to 97%, with a precision and a recall of 98% and 97% respectively, as reported in Table 7.3.

User actions	**Precision**	**Recall**	**F-measure**
refresh home	0.94	0.99	0.96
open contacts	0.97	0.96	0.97
tweet/message	0.97	1.00	0.98
open messages	1.00	0.95	0.97
open twitter	1.00	1.00	1.00
open tweets	1.00	0.95	0.97
other	0.96	0.96	0.96
Average	0.98	0.97	0.97

TABLE 7.3: Classification results of Twitter user actions reached using the Configuration 1.

Performance reached are reported in Figure 7.5. For each user action at least one of the configurations exceeds 96% of accuracy, while the worst configuration has an accuracy in any case higher than 91%.

The user action *open twitter* has accuracy and recall equal to 100%, independently of the configuration set used in the clustering phase. As a consequence, none of examples of the test set have been wrongly classified. Figure 7.6 reports the confusion matrix obtained by considering the Twitter user actions. Three of the six analyzed user actions are correctly classified in more than the 99% of the cases. However, the other three user actions, *open contacts*, *open messages* and *open tweets* are correctly classified in more than 95% of the cases.

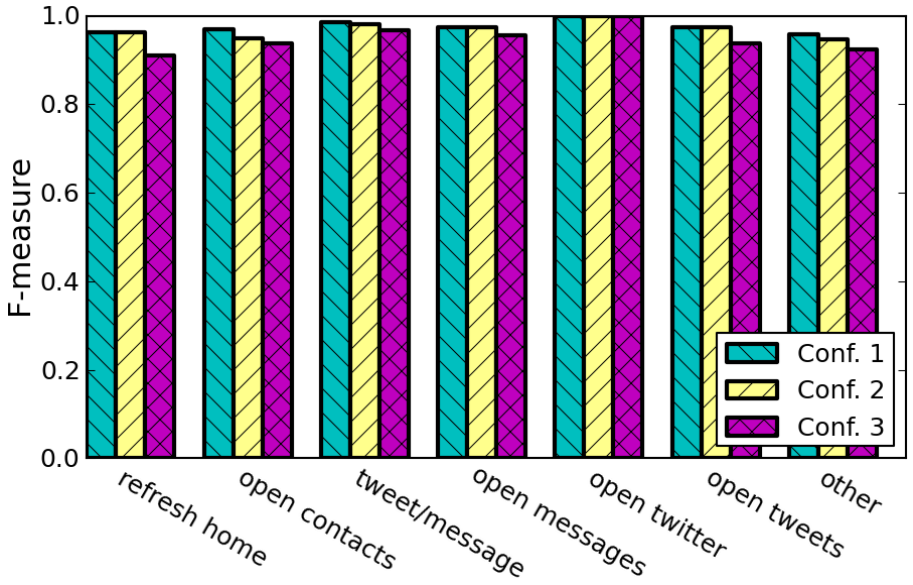

FIGURE 7.5: Classification accuracy of the Twitter user actions.

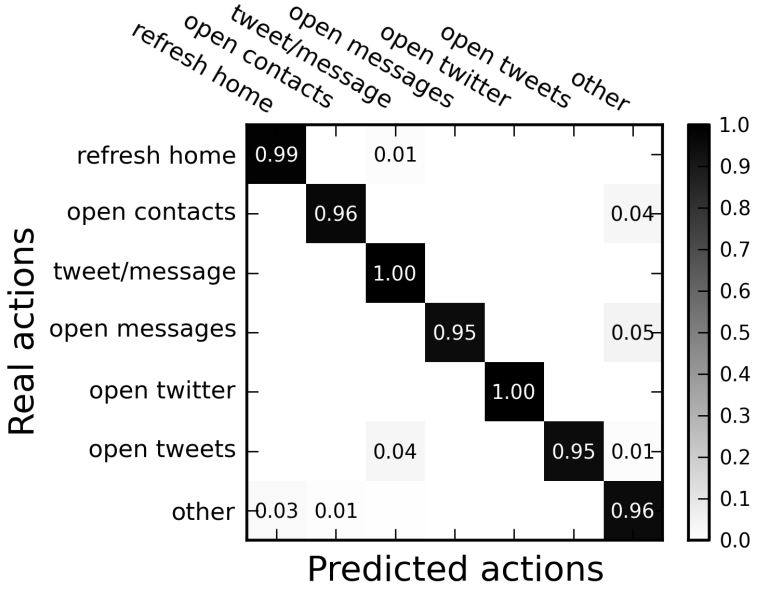

FIGURE 7.6: Twitter user actions confusion matrix for Configuration 1.

Chapter 8

Conclusions

In this thesis, we proposed a framework to analyze encrypted network traffic and to infer which particular actions the user executed on some apps installed on his mobile-phone. We demonstrated that despite the use of SSL or TLS, our traffic analysis approach is an effective tool that an eavesdropper can leverage to undermine the privacy of mobile users. With this tool an attacker may easily learn habits of the target users, and even aggregate data of thousand users in order to gain some commercial or intelligence advantage against some competitor.

We believe that this work will shade a light on the possible attacks that may undermine the user privacy, and that it will stimulate researchers to work on efficient countermeasures that can be adopted also on mobile devices. These countermeasures may require a kind of trade-off between power efficiency and the required privacy level.

To conclude this thesis, in the follow we look ahead, briefly discussing feasibility of countermeasures and suggesting possible future directions for this research.

Possible Countermeasures

Users and service providers might believe that their two parties communications are secure if they use the right encryption and authentication mechanisms. Unfortunately, current secure communication mechanisms limit their traffic encryption actions to the syntax of the transmitted data. The semantic of the communication is not protected in any way [58]. For this reason, it has been possible for example

to develop classifiers for TLS/SSL encrypted traffic that are able to discriminate between applications.

The contribution of this thesis is to investigate to which extent it is feasible to identify the specific user actions that a user is doing on his mobile device, by simply eavesdropping the device's network traffic. While it is out of the scope of this thesis to investigate possible countermeasure to the proposed attack, we discuss in the following some related issues.

One common belief is that simple padding techniques may be effective against traffic analysis approaches. However, it has to be considered that padding countermeasures are already standardized in TLS, explicitly to "frustrate attacks on a protocol that are based on analysis of the lengths of exchanged messages" [59]. Nevertheless, our attack worked against TLS encrypted traffic. More advanced techniques have been proposed in the literature, such as traffic morphing and direct target sampling [31, 60]. However, a recent result showed that none of the existing countermeasures are effective [61]. The intuition is that coarse information is unlikely to be hidden efficiently, and the analysis of these features may still allow an accurate analysis. On the light of these results, we believe it is not trivial to propose effective countermeasures to the attack we shown in this thesis. Indeed, it is intention of the authors to highlight a problem that is becoming even more alarming after the revelation about the mass surveillance programs that are nowadays adopted by governments and nation states.

Future work

In this thesis, we considered network traffic directly related to a user action. A possible future work could be consider the "other side of the coin". This means classify network traffic due to actions made by external users. For example, we would like to know when a user is receiving emails, direct messages or notifications. Classification could also evaluate which clusters are relevant to classify a user action. Starting from those clusters, i.e., flows inside them, we believe a more fine-grained analysis is possible. For example, it would be interesting to assess whether such type of analysis can allow to know the dimension in bytes of content of a post, message or email. We intend to proceed our work trying to classify network traffic generated by user activities on other Android apps. Another possible work could be to adapt our analysis on network traffic generated by iOS devices.

Appendix A

Script log files

In this Appendix, we present examples of log files produced by a script for a specific app. The syntax used in a log file coincides with the one used to define a csv files, with commas that separate a value from another. In the first row, we report the data labels. Each row corresponds to a user action executed by a script. The data labels are:

- **Script ID**: identify the name of the script that execute the user action;
- **Sequence ID**: identifier of the iteration;
- **User action label**: the label that identify the user action executed;
- **Time from epoch**: the number of seconds that have elapsed from Unix epoch;
- **Time from start**: the number of seconds that have elapsed from the start of the current script iteration

In the following, we report log files produced by an iteration of script for Gmail (in Listing A.1), Facebook (in Listing A.2) and Twitter (in Listing A.3). In every reported log files, we consider a sequence of actions that coincide with a single script iteration.

```
Script ID,Sequence ID,User action label,      Time from epoch,Time from start
gmail_01, sequence_1, open gmail,             1392222112.78,  0.00018906593
gmail_01, sequence_1, new mail button a,      1392222223.73,  110.958022118
gmail_01, sequence_1, destination field sel a,1392222254.64,  141.865797043
gmail_01, sequence_1, destination input a,    1392222285.56,  172.789411068
gmail_01, sequence_1, destination done a,     1392222343.52,  230.748980999
gmail_01, sequence_1, subject field select a, 1392222374.47,  261.693037987
gmail_01, sequence_1, subject input a,        1392222405.39,  292.616706133
gmail_01, sequence_1, compose field select a, 1392222452.06,  339.289083958
gmail_01, sequence_1, compose input a,        1392222482.99,  370.210983992
gmail_01, sequence_1, send mail a,            1392222589.14,  476.366657019
gmail_01, sequence_1, new mail button b,      1392222620.08,  507.306937933
gmail_01, sequence_1, destination field sel b,1392222650.98,  538.20936203
gmail_01, sequence_1, destination input b,    1392222681.93,  569.153293133
gmail_01, sequence_1, destination done b,     1392222738.14,  625.368258953
gmail_01, sequence_1, subject field select b, 1392222769.07,  656.289945126
gmail_01, sequence_1, subject input b,        1392222799.95,  687.1730721
gmail_01, sequence_1, compose field select b, 1392222838.74,  725.961789131
gmail_01, sequence_1, compose input b,        1392222869.68,  756.907241106
gmail_01, sequence_1, send mail b,            1392222983.3,   870.520802975
gmail_01, sequence_1, opening menu,           1392223014.18,  901.407958984
gmail_01, sequence_1, open inbox a,           1392223045.04,  932.269609928
gmail_01, sequence_1, opening first email,    1392223075.95,  963.173454046
gmail_01, sequence_1, reply button,           1392223106.85,  994.075664043
gmail_01, sequence_1, mail reply,             1392223137.75,  1024.97882795
gmail_01, sequence_1, reply compose select,   1392223137.75,  1024.97952104
gmail_01, sequence_1, writing reply,          1392223168.66,  1055.8814621
gmail_01, sequence_1, send mail reply,        1392223225.93,  1113.15380192
gmail_01, sequence_1, back to inbox a,        1392223256.85,  1144.07808304
gmail_01, sequence_1, menu selection a,       1392223287.71,  1174.93701696
gmail_01, sequence_1, open prior inbox,       1392223318.62,  1205.84026694
gmail_01, sequence_1, menu selection b,       1392223349.52,  1236.74355507
gmail_01, sequence_1, open important,         1392223380.44,  1267.66457891
gmail_01, sequence_1, menu selection c,       1392223411.34,  1298.56740904
gmail_01, sequence_1, open chats,             1392223442.27,  1329.49088597
gmail_01, sequence_1, menu selection d,       1392223473.21,  1360.43284702
gmail_01, sequence_1, open sent ,             1392223504.13,  1391.35673094
gmail_01, sequence_1, menu selection e,       1392223535.03,  1422.25865006
gmail_01, sequence_1, open draft,             1392223565.96,  1453.18126893
gmail_01, sequence_1, menu selection f,       1392223596.88,  1484.10493302
gmail_01, sequence_1, open outbox,            1392223627.78,  1515.00507498
gmail_01, sequence_1, menu selection g,       1392223658.66,  1545.88897991
gmail_01, sequence_1, open inbox b,           1392223689.65,  1576.870924
gmail_01, sequence_1, search over mail,       1392223720.55,  1607.77297902
gmail_01, sequence_1, writing on search,      1392223751.45,  1638.676121
gmail_01, sequence_1, search button press,    1392223791.0,   1678.22664404
gmail_01, sequence_1, back to inbox b,        1392223821.92,  1709.14818096
gmail_01, sequence_1, menu selection h,       1392223853.09,  1740.31089497
gmail_01, sequence_1, inbox selection c,      1392223883.99,  1771.21476293
gmail_01, sequence_1, first mail select,      1392223914.91,  1802.13945699
gmail_01, sequence_1, delete mail sel,        1392223945.90,  1833.12448096
gmail_01, sequence_1, gmail background,       1392223976.84,  1864.067348
gmail_01, sequence_1, gmail killed,           1392224198.89,  2086.11308813
gmail_01, sequence_1, gmail script ended,     1392224248.94,  2136.16485906
```

Listing A.1: Example of a log file produced by Gmail script for a sequence of actions.

```
Script ID,   Sequence ID,User action label,    Time from epoch,Time from start
facebook_01, sequence_3, open facebook,        1392268976.18,  0.00015401840
facebook_01, sequence_3, menu selection a,     1392269087.19,  111.012023926
facebook_01, sequence_3, replacing menu pos,   1392269118.11,  141.930413008
facebook_01, sequence_3, edit search select,   1392269149.12,  172.943547964
facebook_01, sequence_3, writing search,       1392269180.03,  203.855633974
facebook_01, sequence_3, user friend select,   1392269233.23,  257.054015875
facebook_01, sequence_3, dragging down,        1392269264.18,  287.996669054
facebook_01, sequence_3, write selection,      1392269296.08,  319.899863958
facebook_01, sequence_3, writing to friend,    1392269327.0,   350.822953939
facebook_01, sequence_3, post button select,   1392269438.05,  461.874152899
facebook_01, sequence_3, back to friend page,  1392269468.92,  492.736173868
facebook_01, sequence_3, back to news a,       1392269500.02,  523.839293957
facebook_01, sequence_3, photos news select,   1392269531.08,  554.903454065
facebook_01, sequence_3, back to news b,       1392269561.98,  585.805598974
facebook_01, sequence_3, menu selection b,     1392269593.03,  616.848291874
facebook_01, sequence_3, user profile select,  1392269623.95,  647.771296978
facebook_01, sequence_3, user about select,    1392269654.87,  678.692636013
facebook_01, sequence_3, back to user profile, 1392269685.79,  709.615328074
facebook_01, sequence_3, user photos select,   1392269716.86,  740.678786993
facebook_01, sequence_3, photos album select,  1392269747.8,   771.619953871
facebook_01, sequence_3, back to user profile, 1392269778.76,  802.583379984
facebook_01, sequence_3, friends selection,    1392269809.91,  833.727504015
facebook_01, sequence_3, back to user profile, 1392269840.81,  864.629801989
facebook_01, sequence_3, back to news c,       1392269871.74,  895.563575029
facebook_01, sequence_3, menu selection c,     1392269902.68,  926.496381044
facebook_01, sequence_3, menu message select,  1392269933.55,  957.372344017
facebook_01, sequence_3, conversation select,  1392269964.42,  988.243901968
facebook_01, sequence_3, edit message select,  1392269995.35,  1019.16791487
facebook_01, sequence_3, writing message,      1392270026.26,  1050.07836103
facebook_01, sequence_3, send message select,  1392270127.13,  1150.95598888
facebook_01, sequence_3, menu selection d,     1392270158.04,  1181.86051607
facebook_01, sequence_3, menu news selection,  1392270188.96,  1212.78170896
facebook_01, sequence_3, status selection,     1392270219.88,  1243.70393395
facebook_01, sequence_3, writing status,       1392270250.81,  1274.62744403
facebook_01, sequence_3, status post select,   1392270364.82,  1388.6374929
facebook_01, sequence_3, notif friends select, 1392270395.74,  1419.56058693
facebook_01, sequence_3, closing notif friends,1392270426.56,  1450.38430905
facebook_01, sequence_3, notif message select, 1392270457.44,  1481.25692201
facebook_01, sequence_3, closing notif message,1392270488.33,  1512.148314
facebook_01, sequence_3, notif general select, 1392270519.21,  1543.03004193
facebook_01, sequence_3, closing notif general,1392270550.11,  1573.93431997
facebook_01, sequence_3, facebook background,  1392270580.99,  1604.81465602
facebook_01, sequence_3, facebook killed,      1392270802.98,  1826.80087304
facebook_01, sequence_3, facebook script ended,1392270853.02,  1876.83933306
```

Listing A.2: Example of a log file produced by Facebook script for a sequence of actions.

```
Script ID,  Sequence ID,User action label,     Time from epoch,Time from start
twitter_01, sequence_1, open Twitter,          1391372255.27,  0.00016903877
twitter_01, sequence_1, Homepage select,       1391372366.28,  111.005105019
twitter_01, sequence_1, refresh home,          1391372397.26,  141.986573935
twitter_01, sequence_1, contact select,        1391372429.19,  173.921017885
twitter_01, sequence_1, discover select,       1391372460.16,  204.885387897
twitter_01, sequence_1, me selection,          1391372491.06,  235.791321039
twitter_01, sequence_1, tweets select,         1391372522.0,   266.732984066
twitter_01, sequence_1, back to me a,          1391372552.93,  297.656400919
twitter_01, sequence_1, following select,      1391372584.03,  328.762007952
twitter_01, sequence_1, back to me,            1391372614.96,  359.683928013
twitter_01, sequence_1, followers select,      1391372646.04,  390.767942905
twitter_01, sequence_1, back to me b,          1391372676.94,  421.671257019
twitter_01, sequence_1, direct messages select,1391372708.01,  452.736727953
twitter_01, sequence_1, first message select,  1391372738.94,  483.663450956
twitter_01, sequence_1, edit select,           1391372769.84,  514.566174984
twitter_01, sequence_1, writing message,       1391372800.72,  545.452179909
twitter_01, sequence_1, send select,           1391372855.21,  599.935318947
twitter_01, sequence_1, back to me c,          1391372886.15,  630.880910873
twitter_01, sequence_1, new tweet select,      1391372917.08,  661.804966927
twitter_01, sequence_1, writing tweet,         1391372948.0,   692.729681969
twitter_01, sequence_1, send tweet select,     1391373003.28,  748.007267952
twitter_01, sequence_1, back to home,          1391373034.2,   778.930988073
twitter_01, sequence_1, twitter in background, 1391373065.11,  809.835093975
twitter_01, sequence_1, twitter background,    1391373276.33,  1021.05460191
twitter_01, sequence_1, Twitter killed,        1391373287.28,  1032.00500393
twitter_01, sequence_1, twitter script ended,  1391373337.33,  1082.05790591
```

Listing A.3: Example of a log file produced by Twitter script for a sequence of actions.

Bibliography

[1] Rahad Abir. iphone 5s can track user's every move even after the battery dies, March 2014. URL http://guardianlv.com/2014/03/iphone-5s-can-track-users-every-move-even-after-the-battery-dies/. [Online; accessed April 2nd, 2014].

[2] CNN Staff. Germany: U.s. might have monitored merkel's phone, October 2014. URL http://edition.cnn.com/2013/10/23/world/europe/germany-us-merkel-phone-monitoring/. [Online; accessed April 2nd, 2014].

[3] Roman Schlegel, Kehuan Zhang, Xiao-yong Zhou, Mehool Intwala, Apu Kapadia, and XiaoFeng Wang. Soundcomber: A stealthy and context-aware sound trojan for smartphones. In *Proceedings of the Network and Distributed System Security Symposium*, NDSS '11, 2011.

[4] Sascha Fahl, Marian Harbach, Thomas Muders, Lars Baumgärtner, Bernd Freisleben, and Matthew Smith. Why eve and mallory love android: An analysis of android ssl (in)security. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, 2012.

[5] Martin Georgiev, Subodh Iyengar, Suman Jana, Rishita Anubhai, Dan Boneh, and Vitaly Shmatikov. The most dangerous code in the world: Validating ssl certificates in non-browser software. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, 2012.

[6] Bruno P.S. Rocha, Mauro Conti, Sandro Etalle, and Bruno Crispo. Hybrid static-runtime information flow and declassification enforcement. *Information Forensics and Security, IEEE Transactions on*, Aug 2013.

[7] Yuri Zhauniarovich, Giovanni Russello, Mauro Conti, Bruno Crispo, and Earlence Fernandes. Moses: Supporting and enforcing security profiles on smartphones. *Dependable and Secure Computing, IEEE Transactions on*, 2014.

[8] Guillermo Suarez-Tangil, Juan E. Tapiador, Pedro Peris, and Arturo Ribagorda. Evolution, detection and analysis of malware for smart devices. *IEEE Communications Surveys & Tutorials*, November 2013.

[9] Mauro Conti, Nicola Dragoni, and Sebastiano Gottardo. Mithys: Mind the hand you shake - protecting mobile devices from ssl usage vulnerabilities. In *ESORICS Workshop on Security and Trust Management*, 2013.

[10] Claudio A. Ardagna, Mauro Conti, Mario Leone, and Julinda Stefa. An anonymous end-to-end communication protocol for mobile cloud environments. *Services Computing, IEEE Transactions on*, 2014.

[11] Tim Stöber, Mario Frank, Jens Schmitt, and Ivan Martinovic. Who do you sync you are?: Smartphone fingerprinting via application behaviour. In *Proceedings of the Sixth ACM Conference on Security and Privacy in Wireless and Mobile Networks*, WiSec '13, 2013.

[12] Nino Vincenzo Verde, Giuseppe Ateniese, Emanuele Gabrielli, Luigi Vincenzo Mancini, and Angelo Spognardi. No nat'd user left behind: Fingerprinting users behind nat from netflow records alone. Technical report, feb 2014.

[13] Erika Chin, Adrienne Porter Felt, Vyas Sekar, and David Wagner. Measuring user confidence in smartphone security and privacy. In *Proceedings of the Eighth Symposium on Usable Privacy and Security*, SOUPS '12, 2012.

[14] Adrienne Porter Felt, Matthew Finifter, Erika Chin, Steve Hanna, and David Wagner. A survey of mobile malware in the wild. In *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, SPSM '11, 2011.

[15] Thomas Eder, Michael Rodler, Dieter Vymazal, and Marcus Zeilinger. Ananas - a framework for analyzing android applications. In *Availability, Reliability and Security (ARES), 2013 Eighth International Conference on*, Sept 2013.

[16] Sanae Rosen, Zhiyun Qian, and Z. Morely Mao. Appprofiler: A flexible method of exposing privacy-related behavior in android applications to end users. In *Proceedings of the Third ACM Conference on Data and Application Security and Privacy*, CODASPY '13, 2013.

[17] Zhemin Yang, Min Yang, Yuan Zhang, Guofei Gu, Peng Ning, and X. Sean Wang. Appintent: Analyzing sensitive data transmission in android for privacy

leakage detection. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, 2013.

[18] Xuetao Wei, Lorenzo Gomez, Iulian Neamtiu, and Michalis Faloutsos. Profiledroid: Multi-layer profiling of android applications. In *Proceedings of the 18th Annual International Conference on Mobile Computing and Networking*, Mobicom '12, 2012.

[19] Adrienne Porter Felt, Elizabeth Ha, Serge Egelman, Ariel Haney, Erika Chin, and David Wagner. Android permissions: User attention, comprehension, and behavior. In *Proceedings of the Eighth Symposium on Usable Privacy and Security*, SOUPS '12, 2012.

[20] Peter Hornyack, Seungyeop Han, Jaeyeon Jung, Stuart Schechter, and David Wetherall. These aren't the droids you're looking for: Retrofitting android to protect data from imperious applications. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, 2011.

[21] William Enck, Peter Gilbert, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth. Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, 2010.

[22] Alastair R. Beresford, Andrew Rice, Nicholas Skehin, and Ripduman Sohan. Mockdroid: Trading privacy for application functionality on smartphones. In *Proceedings of the 12th Workshop on Mobile Computing Systems and Applications*, HotMobile '11, 2011.

[23] Chiung Ching Ho, C. Eswaran, Kok-Why Ng, and June-Yee Leow. An unobtrusive android person verification using accelerometer based gait. In *Proceedings of the 10th International Conference on Advances in Mobile Computing & Multimedia*, MoMM '12, 2012.

[24] Mauro Conti, Irina Zachia-Zlatea, and Bruno Crispo. Mind how you answer me!: Transparently authenticating the user of a smartphone when answering or placing a call. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '11, 2011.

[25] Kamil Majdanik, Cristiano Giuffrida, Mauro Conti, and Herbert Bos. I sensed it was you: Authenticating mobile users with sensor-enhanced keystroke dynamics. In *Proceedings of the 11th Conference on Detection of Intrusions and Malware & Vulnerability Assessment*, DIMVA '14, 2014.

[26] Yeon-sup Lim, Hyun-chul Kim, Jiwoong Jeong, Chong-kwon Kim, Ted "Taekyoung" Kwon, and Yanghee Choi. Internet traffic classification demystified: On the sources of the discriminative power. In *Proceedings of the 6th International COnference*, Co-NEXT '10, 2010.

[27] Hyunchul Kim, KC Claffy, Marina Fomenkov, Dhiman Barman, Michalis Faloutsos, and KiYoung Lee. Internet traffic classification demystified: Myths, caveats, and the best practices. In *Proceedings of the 2008 ACM CoNEXT Conference*, CoNEXT '08, 2008.

[28] Andrew W. Moore and Denis Zuev. Internet traffic classification using bayesian analysis techniques. *SIGMETRICS Perform. Eval. Rev.*, 33(1), June 2005.

[29] Thuy T. T. Nguyen, Grenville Armitage, Philip Branch, and Sebastian Zander. Timely and continuous machine-learning-based classification for interactive ip traffic. *IEEE/ACM Trans. Netw.*, December 2012.

[30] Laurent Bernaille, Renata Teixeira, and Kave Salamatian. Early application identification. In *Proceedings of the 2006 ACM CoNEXT Conference*, CoNEXT '06, 2006.

[31] Charles V. Wright, Lucas Ballard, Scott E. Coull, Fabian Monrose, and Gerald M. Masson. Spot me if you can: Uncovering spoken phrases in encrypted voip conversations. In *Proceedings of the 2008 IEEE Symposium on Security and Privacy*, SP '08, 2008.

[32] Dawn Xiaodong Song, David Wagner, and Xuqing Tian. Timing analysis of keystrokes and timing attacks on ssh. In *Proceedings of the 10th Conference on USENIX Security Symposium - Volume 10*, SSYM '01, 2001.

[33] Gabriel Gómez Sena and Pablo Belzarena. Early traffic classification using support vector machines. In *Proceedings of the 5th International Latin American Networking Conference*, LANC '09, 2009.

[34] Pedro Casas, Johan Mazel, and Philippe Owezarski. Minetrac: Mining flows for unsupervised analysis & semi-supervised classification. In *Proceedings of the 23rd International Teletraffic Congress*, ITC '11, 2011.

[35] Anthony Mcgregor, Mark Hall, Perry Lorier, and James Brunskill. Flow clustering using machine learning techniques. In *PAM '04*, 2004.

[36] Gabriel Gómez Sena and Pablo Belzarena. Statistical traffic classification by boosting support vector machines. In *Proceedings of the 7th Latin American Networking Conference*, LANC '12, 2012.

[37] Fan Zhang, Wenbo He, Xue Liu, and Patrick G. Bridges. Inferring users' online activities through traffic analysis. In *Proceedings of the Fourth ACM Conference on Wireless Network Security*, WiSec '11, 2011.

[38] Thomas Karagiannis, Konstantina Papagiannaki, and Michalis Faloutsos. Blinc: Multilevel traffic classification in the dark. In *Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '05, 2005.

[39] Manuel Crotti, Maurizio Dusi, Francesco Gringoli, and Luca Salgarelli. Traffic classification through simple statistical fingerprinting. *SIGCOMM Comput. Commun. Rev.*, January 2007.

[40] Jun Zhang, Chao Chen, Yang Xiang, and Wanlei Zhou. Robust network traffic identification with unknown applications. In *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, ASIA CCS '13, 2013.

[41] Richard Atterer, Monika Wnuk, and Albrecht Schmidt. Knowing the user's every move: User activity tracking for website usability evaluation and implicit interaction. In *Proceedings of the 15th International Conference on World Wide Web*, WWW '06, 2006.

[42] Fabian Schneider, Anja Feldmann, Balachander Krishnamurthy, and Walter Willinger. Understanding online social network usage from a network perspective. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference*, IMC '09, 2009.

[43] Fabrício Benevenuto, Tiago Rodrigues, Meeyoung Cha, and Virgílio Almeida. Characterizing user navigation and interactions in online social networks. *Inf. Sci.*, July 2012.

[44] Scott Renfro. Secure browsing by default, 2013. URL https://www.facebook.com/notes/facebook-engineering/secure-browsing-by-default/10151590414803920. [Online; accessed April 2nd, 2014].

[45] Luis Cipriani. Restricting api.twitter.com to ssl/tls traffic, 2014. URL https://dev.twitter.com/discussions/24239. [Online; accessed April 2nd, 2014].

[46] Jean-François Raymond. Traffic analysis: Protocols, attacks, design issues, and open problems. In *International Workshop on Designing Privacy Enhancing Technologies: Design Issues in Anonymity and Unobservability*, 2001.

[47] Oliver Berthold, Hannes Federrath, and Marit Köhntopp. Project anonymity and unobservability in the internet. In *Proceedings of the Tenth Conference on Computers, Freedom and Privacy: Challenging the Assumptions*, CFP '00, 2000.

[48] Marc Liberatore and Brian Neil Levine. Inferring the source of encrypted http connections. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, 2006.

[49] Dominik Herrmann, Rolf Wendolsky, and Hannes Federrath. Website fingerprinting: Attacking popular privacy enhancing technologies with the multinomial naive-bayes classifier. In *Proceedings of the 2009 ACM Workshop on Cloud Computing Security*, CCSW '09, 2009.

[50] Andriy Panchenko, Lukas Niessen, Andreas Zinnen, and Thomas Engel. Website fingerprinting in onion routing based anonymization networks. In *Proceedings of the 10th Annual ACM Workshop on Privacy in the Electronic Society*, WPES '11, 2011.

[51] Shuaifu Dai, Alok Tongaonkar, Xiaoyin Wang, Antonio Nucci, and Dawn Song. Networkprofiler: Towards automatic fingerprinting of android apps. In *Proceedings of the 2013 IEEE INFOCOM*, 2013.

[52] Alok Tongaonkar, Shuaifu Dai, Antonio Nucci, and Dawn Song. Understanding mobile app usage patterns using in-app advertisements. In *Proceedings of the 14th International Conference on Passive and Active Measurement*, PAM '13, 2013.

[53] Hossein Falaki, Dimitrios Lymberopoulos, Ratul Mahajan, Srikanth Kandula, and Deborah Estrin. A first look at traffic on smartphones. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, 2010.

[54] Meinard Müller. *Information Retrieval for Music and Motion*. Springer-Verlag New York, Inc., 2007.

[55] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning (2nd ed.)*. Springer Series in Statistics. Springer New York Inc., 2009.

[56] Harry Zhang. The optimality of naive bayes. *A A*, 2004.

[57] Leo Breiman. Random forests. *Machine Learning*, October 2001.

[58] Balachander Krishnamurthy. Privacy and online social networks: Can colorless green ideas sleep furiously? *IEEE Security and Privacy*, May 2013.

[59] Travis Dierks and Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008.

[60] Charles V. Wright, Scott E. Coull, and Fabian Monrose. In *Proceedings of the Network and Distributed System Security Symposium*, NDSS '09, 2009.

[61] Kevin P. Dyer, Scott E. Coull, Thomas Ristenpart, and Thomas Shrimpton. Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, SP '12, 2012.