Master Thesis

Who do you sync you are?

Smartphone Fingerprinting based on Application Behaviour

Tim Stöber

University of Kaiserslautern
Department of Computer Science
Distributed Computer Systems Lab

Examiner: Prof. Jens Schmitt
Dr. Ivan Martinovic

Abstract

The overall network traffic patterns generated by today's smartphones result from the typically large and diverse set of installed applications. In addition to the traffic initiated through user interactions, most applications generate characteristic traffic from their background activities, such as periodic update requests, keeping alive connections, or server synchronisations.
Although the encryption of transmitted data in 3G/UMTS networks prevents an eavesdropper from analysing the content, periodic traffic patterns leak as side-channel information, e.g. in the time and amount of transmitted data domain. In this work, we extract such side-channel information features from network traffic and evaluate whether they can be used to reliably identify a smartphone. By analysing the background traffic from the most popular applications, such as Facebook App, WhatsApp, Skype, Dropbox, and others, we demonstrate that within 15 minutes of monitoring traffic, an eavesdropper can identify a smartphone based on its behavioural fingerprint with a success probability of 90%.

Der Internetverkehr moderner Smartphones setzt sich aus Übertragungen einzelner installierter Anwendungen zusammen. Neben dem Datenverkehr welcher durch Benutzerinteraktion initiiert wird, verursachen diverse Anwendungen regelmäßige Internetkommunikation in Hintergrundprozessen, ohne dass die jeweilige App gerade aktiv im Vordergrund ist. Darunter fallen z.B. periodische Updates, Server Synchronisationen oder Aufrechterhalten von Verbindungen.
Obwohl ein potentieller Angreifer aufgrund von Verschlüsselungsmaßnahmen in 3G/UMTS Netzwerken gehindert ist Dateninhalte zu analysieren, ist es dennoch möglich sogenannte „Side-channel“ Informationen aus Verkehrsmustern zu beziehen. In dieser Arbeit werden solche Side-channel Informationen (aus der Zeit- und Datendomäne) aus dem Datenverkehr extrahiert und anschließend evaluiert, ob diese sich eignen Smartphones in zuverlässiger Art und Weise voneinander zu unterscheiden. Durch Analyse des im Hintergrund erzeugten Datenverkehrs der populärsten Applikationen wie Facebook, WhatsApp, Skype, Dropbox u.a. demonstrieren wir, dass ein Angreifer mit nur 15 Minuten mitgeschnittenem Verkehr ein Smartphone, basierend auf dessen verhaltensbezogenen Fingerabdruck, mit einer Erfolgswahrscheinlichkeit von 90% identifizieren kann.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

Over the last decade, smartphones became omnipresent. According to [1], 419 million devices have been purchased all over the world during the second quarter of 2012, and half of all US mobile subscribers own a smartphone, with an upward tendency [2]. The main reasons for such a popularity of smartphones are the overall improvement in performance, battery life, and the decreasing price of mobile Internet access over 3G radio networks. In particular, the frequent Internet access is crucial for the success of application markets and a high number of downloaded applications (Apps), such as Email clients, Facebook, WhatsApp, Skype, or Dropbox. Most of these Apps generate traffic, which is not only initiated by the user through interaction, but also from the App itself to maintain the most current state, synchronise, or keep connections alive. For example, Figure 1.1 shows results of our measurements of network traffic transmitted to and from different smartphones for a period of 24 hours (more detail on experimental design is given in Chapter 4).

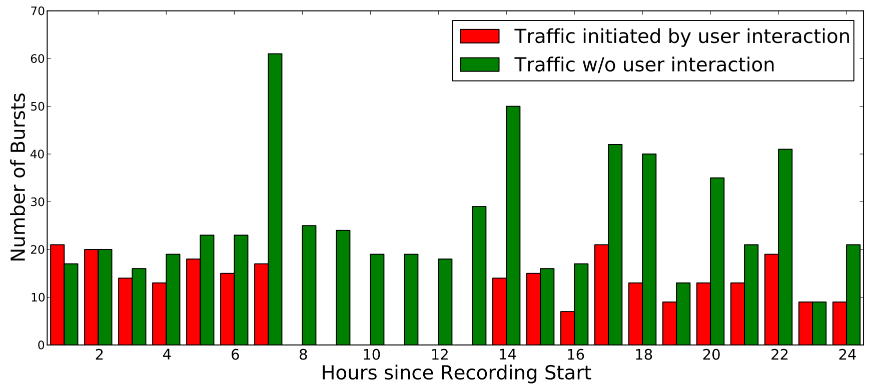

Figure 1.1: Proportions of interactive- and non-interactive smartphone traffic (24 hours). Only about 30% of transmissions are user initiated.

We identified that only 30% of the overall smartphone traffic can be attributed to user interactions (we refer to it as *interactive traffic*), and 70% of the traffic belongs to background activities generated by various installed Apps. Many of such background activities result in characteristic traffic patterns, especially in the time domain and the amount of transmitted data. In addition, the generated traffic highly depends on the multitude of installed applications and their personal configuration.

The 3G/UMTS radio access technology implements encryption at the data link layer to guarantee confidentiality in the presence of a wireless eavesdropper. Yet, the resulting application-dependent traffic patterns may still pose a privacy risk. An attacker might be able to distinguish different smartphones only by analysing side-channel information of encrypted traffic. Hence, this motivates the main question of this work: is it possible to identify a smartphone based on the traffic behaviour of the installed applications, arising from their background activities? Importantly, we assume that an eavesdropper is completely agnostic to the content of the traffic and that security services of the current UMTS radio access network remain unaffected.

In our scenario, the adversary is a passive eavesdropper inherently able to capture encrypted wireless 3G/UMTS data from a victim's smartphone and using that traffic to extract fingerprints. The fingerprints would then allow him to identify the devices afterwards and make a point of whether a particular smartphone is present within a certain UMTS radio cell. While the granularity of such UMTS cell-based tracking does not provide a very fine-grained physical position of the smartphone, it might still reveal important privacy information and help launching more sophisticated attacks such as the detection if the user is at home or at the workplace. Hence, to be able to better assess risks of identifying a smartphone through analysing side-channel information of its background traffic, the major contribution of this thesis is to investigate the following research questions:

- How discriminative are the features of the smartphone traffic generated by applications' background activities?
- Are individual combinations of installed Apps and their personalised configurations sufficient to distinguish between different smartphones?
- How robust are such fingerprints in the presence of artefacts (such as those arising from interactive traffic initiated by users)?
- How long does it take to accurately identify a smartphone?

1.2 Overview

The roadmap of this thesis is as follows:

In **Chapter 2** we will provide insights into the 3G/UMTS physical layer and discuss the feasibility of a realistic UMTS eavesdropper. Moreover, we will have a closer look at smartphone traffic and introduce some supervised machine learning techniques which will be applied for the fingerprinting and identification procedures.

Data acquisition is primarily achieved by capturing real-world traffic from 15 top downloaded Apps. All in all, our experimental designs are based on three different datasets, which are presented in **Chapter 3**.

Chapter 4 addresses the generation as well as the identification of the most discriminative features of fingerprints. A step-by-step process describes the construction of a fingerprint out of a piece of recorded smartphone traffic. Subsequently, the quality of single fingerprints is estimated by identifying important contributing factors.

After being in possession of a fingerprint, **Chapter 5** briefly introduces the process of smartphone identification and its prerequisites.

In an experimental section (**Chapter 6**), we analyse the accuracy of identifying smartphones based on their fingerprints, also depending on the amount of available traffic. Furthermore, we examine how long it would take an attacker to attain reliable identification, as well as the effects of traffic caused by user interaction.

Finally, the thesis will be concluded and open topics as well as future work will be discussed in **Chapter 7**.

1.3 Related Work

There has been much research in the area of device fingerprinting. In most cases side-channel information, such as hardware and manufacturing inconsistencies, or differences in driver implementations were exploited as a discriminative component. Probably one of the first publications in this direction was [18] by Kohno et al. in 2005. They introduced a mechanism to remotely fingerprint and identify devices based on their clock skews, which were in turn based on information from TCP and ICMP time stamps. In [13], the authors succeeded in fingerprinting and identifying wireless device drivers on the basis of a statistical analysis of a device's interarrival rate of IEEE 802.11 probe request frames. Since the standard does not provide a concrete value for the scanning intervals of these management frames, distinct drivers tend to differ in their implementations. Based on this work, Loh et al. extended the discriminability of this feature to be even capable of distinguishing between single devices instead of device drivers [9].
In [14, 22, 7, 6], authors exploit manufacturing inconsistencies and hardware imperfections that have effect on the resulting transmission signal in order to differentiate between single entities.
In addition, one can find a very detailed and systematic review of physical-layer identification systems and state-of-the-art techniques in [8].
All previously mentioned approaches are aiming at remote identification of devices, however, the fingerprint does not include any form of application behaviour.

In contrary, researchers began utilising biometrics for identification and authentication purposes. For example making use of mouse movements [28] or a combination of mouse and keystroke dynamics [5], profiles are created in order to accomplish user verification and intrusion detection. In [12], the authors employ the user's touch behaviour on smartphone screens in order to obtain a continuous and implicit authentication. However, these forms of identification that exploit users' behavioural characteristics as side-channel information are implemented on the device itself and cannot be achieved remotely.

Another related area of research includes various approaches to traffic classification. Some of them utilise transport layer statistics like packet size, connection duration and ratio of bytes sent in each direction, combined with unsupervised machine learning algorithms [10, 26]. Even though the features are solely from the time and byte dimension, both techniques still rely on the notion of flow or connection, respectively, implying that an adversary needs at least an insight into transport layer headers. Yet, an UMTS eavesdropper is unable to include such information into the traffic analysis. In [27], the authors use a packet-level classification, only resorting to link layer features, which makes their approach applicable even under payload encryption. Similarly to our work, their methodology uses supervised learning algorithms like SVM and neural networks. However, their application and network environment is not focused on wireless networks.

Other lines of work interested in detecting network applications, discusses several other information that can be leaked through traffic analysis, e.g., visited web pages [19, 21], language and spoken phrases [24, 25], or watched videos [20]. As opposed to these purposes, our approach applies traffic classification in order to deduce the actual user.

2 Foundation

2.1 UMTS Air Interface

2.1.1 Wideband Code Division Multiple Access (W-CDMA)

Unlike GSM, UMTS does not rely on time multiplexing anymore. Instead, 3G technology is based on code division multiple access (CDMA), in particular W-CDMA, such that every communication channel between two parties is assigned an orthogonal *channelisation code*. This allows several transmissions to proceed on the same frequency and at the same time.
Since the deployment of channelisation codes increases the bandwidth by literally spreading the signal in terms of frequency, they are also referred to as *spreading codes*. The length of the code, also called spreading factor, is measured in chips and can be variable, however, the spread data rate always has to be $3.84 Mchips/s$. Supposed we have a $120 kbs$ data stream, we would have to apply a spreading factor of 32 in order to end up with the prescribed chipping rate of $3.84 Mchips/s$ ($32 \frac{chips}{bit} \cdot 120 kbs = 3.84 Mchips/s$). Since every bit of the data stream is multiplied by the spreading code, the latter's length always relates to the increase factor of bandwidth.
Spreading codes have different roles on up- and downlink. On the uplink, they are used to distinguish between several channels of one single user equipment (UE), on the downlink, different connections from the base station (Node-B) to the UEs are separated. As already mentioned, all applied codes have to be orthogonal in the sense of their scalar product, whereas the 0-bits of the code vectors are substituted by -1 during the multiplication. Being in possession of the utilised spreading code, the receiver is in a position to extract its data from the interfered signal. This principal perfectly works for the point to multipoint transmission from the Node-B to the user devices. However, since the UEs are not synchronised, it is likely to happen that two different cell users UE_1 and UE_2 start their transmissions shifted in time such that their channelisation codes lose the orthogonality property.
This is where the *scrambling code* comes in. In order to distinguish single UEs from each other, every user is assigned an individual scrambling code that is multiplied chip by chip with the spread chipping sequence. The scrambling itself does not introduce a further increase of bandwidth but is just for orthogonal coding.
Even though we deal with a synchronous point to multipoint transmission on the downlink, there might be other Node-Bs in reception range, again resulting in time

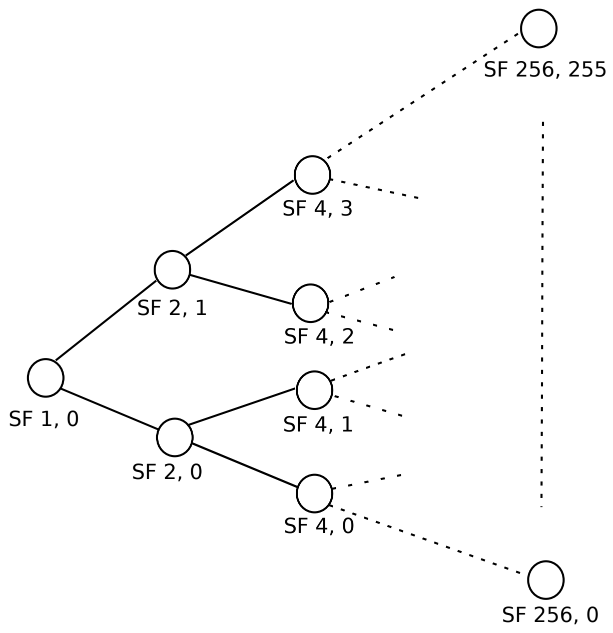

Figure 2.1: Illustration of an OVSF code tree. The left number in the label represents the length of the corresponding code (spreading factor), the right number is an enumerating index.

shifts of the actual orthogonal channelisation codes. Hence, in order to differentiate distinct cells, a scrambling code has to be introduced on the downlink as well. The following two subsections explicitly consider the characteristics and assignments of spreading- and scrambling codes on the up- and downlink.

Uplink

The channelisation codes for uplink spreading are picked out of a set of orthogonal vector spreading factor codes (OVSF) which can be conceived of as a tree shape, comparable to Figure 2.1. The tree consists of 511 labelled nodes, whereas the root is inscribed by a binary one. Assuming an arbitrary node is labelled by the bit sequence X, its child nodes will be labelled by the double lengths sequences X,X (right child) and X,-X (left child). However, not all of the 511 may be used simultaneously. Choosing a random node n from the tree results in a blocking of all codes from n's sub-trees as well as all codes on the path from n to the root node. On the uplink, the spreading factors are defined as all powers of two, ranging from 4 to 256. Thus, only 508 of the 511 codes can be applied but at most 256 of them in parallel. Now, essentially every physical channel is assigned a different channelisation code, whereas some of them have a fixed pre-assigned code (e.g. DPCCH utilises SF 256,0 [17]).

In terms of the scrambling code, there exist two different variants. Either a 256 chip short scrambling code, which is applied if the Node-B is equipped with an advanced receiver (e.g. multiuser detector), or a long scrambling code, consisting of 38400 chips, that exactly matches the length of one radio frame (10ms). Due to the phase modulation, there is a need of two scrambling codes, one for the real- and one for the imaginary bitstream [3]. Both sequences are generated using a 25-bit shift register seeded with a 24-bit random bit string that is passed to the UE by upper layers from the responsible radio network controller (RNC). The resulting long scrambling codes are also referred to as Gold codes.

Downlink

The scrambling codes on the downlink are created in a similar way as on the uplink, however, instead of 25 bits, only 18 bits are used as a shift register seed. Moreover, out of the $2^{18} - 1$ possible outcomes, only 8192 are actually applied. The overall set is divided into 512 primary codes, each of them having an associated set of 15 secondary scrambling codes. Furthermore, there exist right and left alternate scrambling codes that can be computed on the basis of the primary code. For further details on their computations, we point to the technical specification 3GPP TS 25.223 [4].

With regard to spreading, OVSF codes are utilised similarly to the uplink scenario, however this time with spreading factors from 4 to 512. For each scrambling code a cell is using (48 at most), there is one code tree at hand, whereas due to pre-assignments of particular physical channels, only 986 out of 1020 available codes remain unoccupied [17].

2.1.2 On Capturing Side-Channel Information of 3G Traffic

In the first place, we assume that the attacker is located within the range of transmissions, such that the inherent broadcast nature of the wireless medium allows him to eavesdrop on UMTS physical signals. Hence, our main assumption is that the attacker is able to demodulate and demultiplex the physical layer signal and measure side-channel information such as the amount of transmitted data and timing information. In order to acquire such side-channel information, the attacker must extract the users' data streams from the superimposed W-CDMA signal on the air interface. In this section, we briefly discuss technical requirements and the complexity to gather side-channel information without the possession of spreading and scrambling codes used in UMTS.

The users' data (payload) is transmitted over so-called Dedicated Physical Data Channels (DPDCH), and its correct demodulation requires the knowledge of scrambling and spreading codes as specified in [4]. However, none of these coding techniques were designed for security reasons since the security objectives are offered by the higher layers of the UMTS network stack. In particular, both spreading and scrambling codes can be "brute-forced" as their search space is not large and the assignment of channelisation codes on the uplink is almost completely specified in [4]. For instance, assuming one Node-B using one primary scrambling code for its cell, there are less than 1000 available spreading codes that could be employed after deducting codes for reserved common channels [17]. The scrambling code is more expensive to find, as it is generated by 18 bit (downlink) and 24 (uplink) seeded shift registers, respectively. Yet, none of these lengths present a significant computational burden for an adversary. Moreover, the Node-B's scrambling code for the downlink is even automatically determinable by the smartphone's cell search procedure on the Common Pilot Channel (CPICH). Table 2.1 shows a cost estimate for obtaining the required codes in the downlink and uplink cases.

	Scrambling	**Spreading**
Up	2^{24} possibilities	max. 7 possibilities
Down	2^{18} possibilities (*determinable on CPICH*)	max. 1000 possibilities

Table 2.1: Cost estimate for attaining scrambling- and spreading codes. Determination of Node-B's scrambling code is less expensive in contrast to the UEs'.

As it can be seen, it is much easier for an attacker to get hold of downlink traffic. In our experiments, we therefore also investigate how the availability of downlink-only traffic affects the success probability of correctly identifying a smartphone.

2.2 Burstiness of Smartphone Traffic

Taking a closer look at smartphone traffic, one can observe alternating idle periods followed by short peaks of incoming and outgoing data transfers. This behaviour, which we refer to as burstiness, is illustrated in Figure 2.2 and has also been identified and described by Falaki et al. in [11]. The x-axis shows the timeline and the y-axis indicates the amount of transmitted and received bytes. A single *burst*, represented by the peaks of the black curve, can consist of one or more transmissions, which are sequences of packets that are semantically connected to each other like packets from the same TCP connection. Such a transmission either relates to a certain application causing traffic in background or arose due to a user interaction, such as the request of a website or sending an email. Figure 2.3 depicts a simplified graphical illustration of a single burst. The red and green bars represent outgoing respectively incoming packets, whereas their height is in place of the lengths in bytes. Further information on how bursts are extracted from the traffic and how they are characterised in terms of features can be found in Sections 4.2 and 4.3.

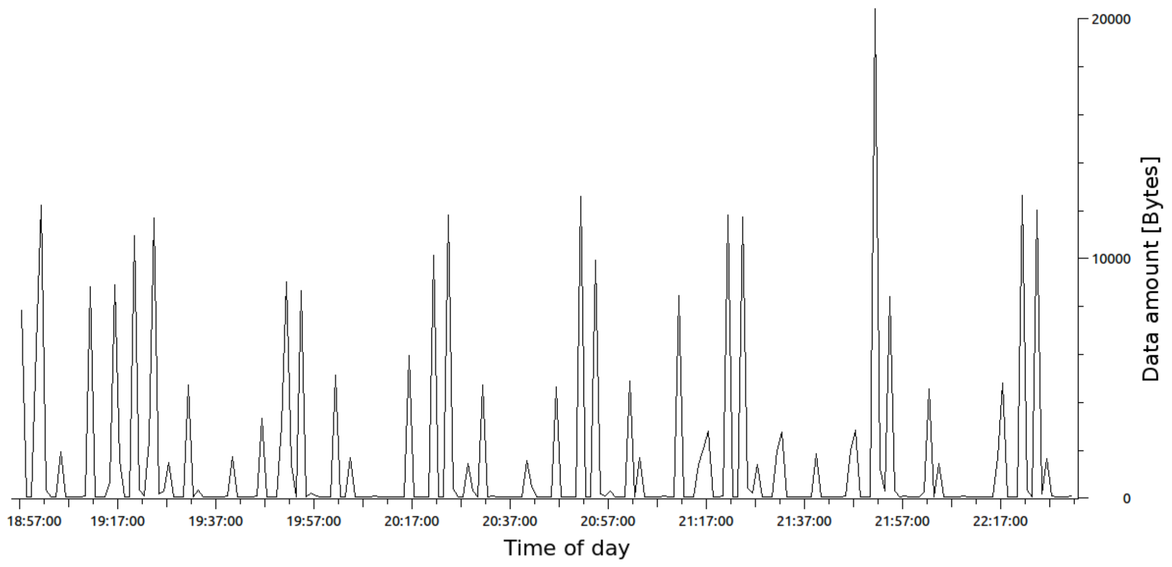

Figure 2.2: 3.5 hours of smartphone traffic. Idle phases alternating with transmissions illustrate the burstiness. As it can be observed, similar burst patterns occur in regular intervals.

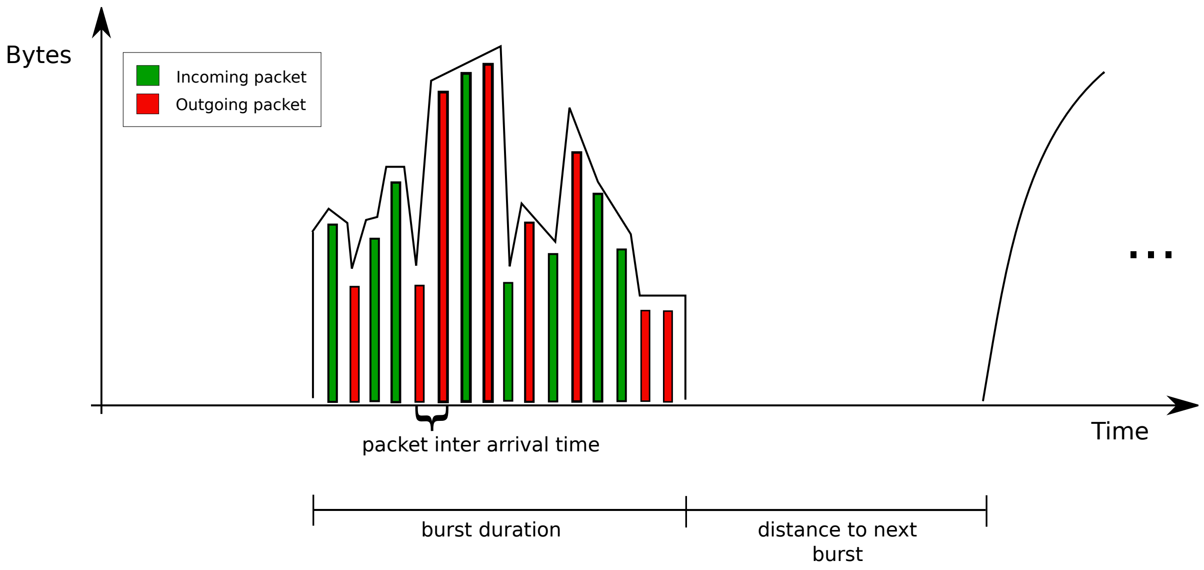

Figure 2.3: Simplified graphical representation of a burst.

2.3 Machine Learning

Since our future goal is to make a point of whether traffic arose from a particular user or not, we face the task of solving a classification problem. Therefore, we apply algorithms from the field of supervised machine learning. In terms of classification, supervised machine learning essentially means the construction of a classification function (classifier) on the basis of labelled training data. The overall aim is to infer a classifier that afterwards assigns class labels to unlabelled observations as precisely as possible. In the following two subsections, we will introduce two specific implementations of classification algorithms.

2.3.1 k-Nearest Neighbour Algorithm

The k-nearest neighbour algorithm (kNN) is based on a very comprehensible approach. It does not require an explicit training phase, since all observations are simply stored in the feature space in terms of their feature vectors and labels. The dimension of the feature space corresponds to the length of the feature vectors that describe the observations.
In order to classify an unlabelled observation, the algorithm computes the k nearest training samples by means of a specified distance metric (e.g. Euclidean distance or city block distance). The new label is then chosen by the majority vote of the k nearest neighbors. Thus, setting the parameter k is a crucial step. A small k makes the outcome prone to noise, however, a very large k might exceed the class boundaries and additionally results in a higher computational effort. The example in Figure 2.4 illustrates the importance of the parameter k. The shapes and colors of the geometrical forms represent their class membership, whereas the red circle marked with a cross is the new observation that has to be classified. Choosing $k = 3$ results in a falsified labelling due to the presence of noise, whereas a too large value ($k = 35$) exceeds the class boundaries of the red circles and also leads to a wrong classification.

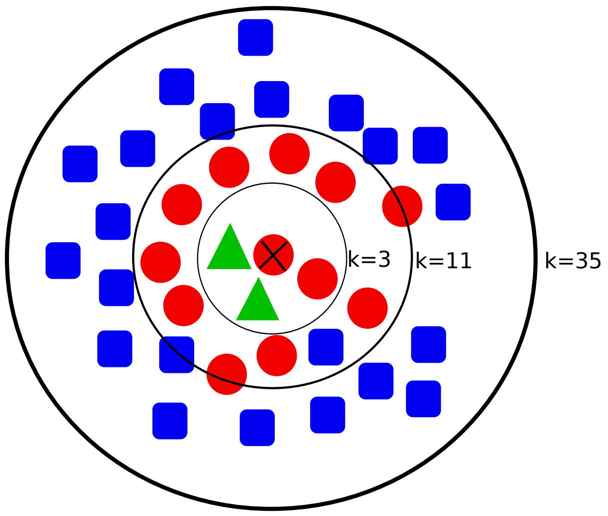

Figure 2.4: kNN classification example, illustrating the importance of choosing the right parameter setting for k.

In the case of large data sets, the algorithm becomes inefficient. Not only because all observations have to be stored, but also in terms of computational expense, since distances have to be calculated for every single available training sample.

2.3.2 Support Vector Machine

Unlike kNN, which can naturally be used to solve multi-classification problems, the Support Vector Machine (SVM) is a binary classifier. Instead of storing all training observations, the SVM generalises from the training data by computing one or more maximally wide hyperplanes that linearly separate the two classes from each other. As a result, only the hyperplanes have to be remembered and can be described in terms of the vectors that are closest to them, the so-called support vectors. By applying the so-called kernel trick, even non-linearly separable classes can be split. This is realised by mapping the observations into a high-dimensional feature space where they become separable. Figure 2.5 shows an easy and illustrative example of two-dimensional data separated by one maximally thick hyperplane. The corresponding support vectors are marked by a cross. In contrary, Figure 2.6 depicts a non-linearly separable set of feature vectors (left image) which is transfered to a higher dimension in order to become linearly separable (right image). The subsequent classification is simply decided by the side of the hyperplane on which the new observation is located.
A more detailed explanation of support vector machines can be found in [23].

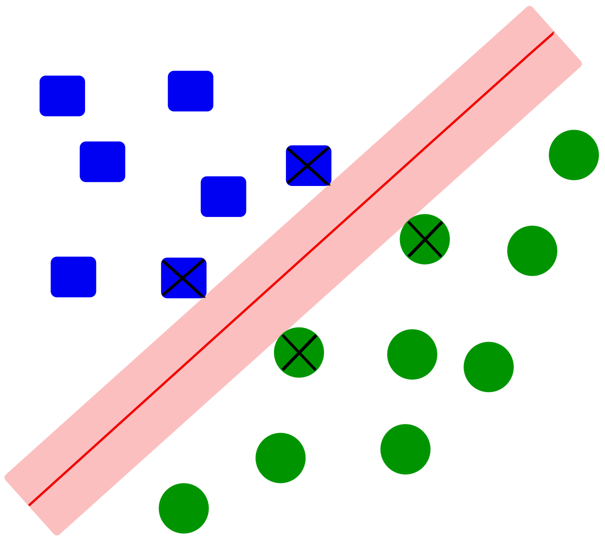

Figure 2.5: Separation of two-dimensional data by a maximally wide hyperplane. Support vectors are marked by a cross.

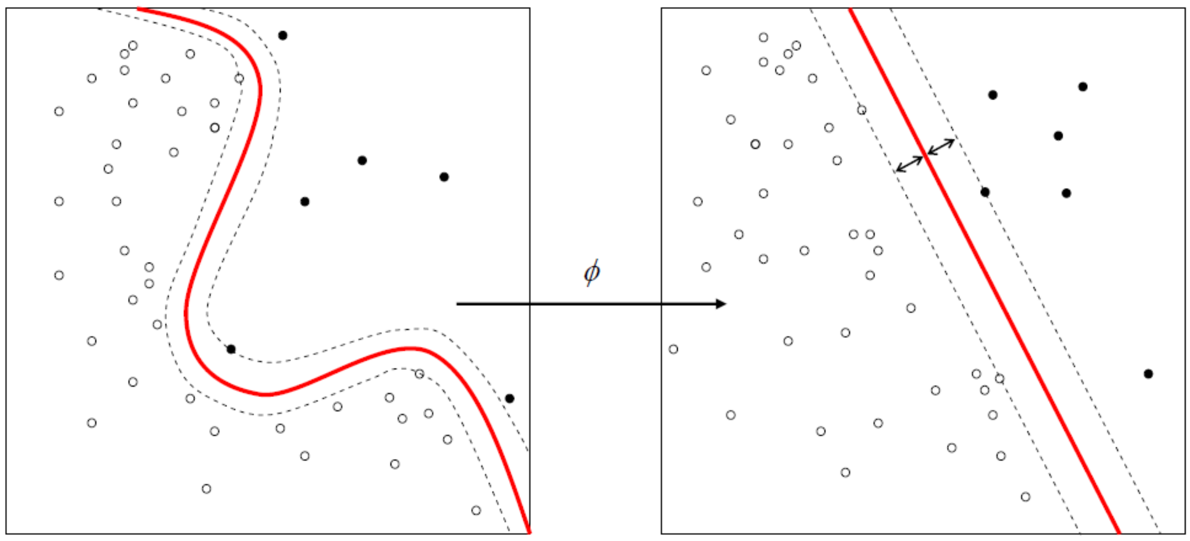

Figure 2.6: Graphical illustration of the kernel trick in order to transfer feature vectors into a higher dimension with an existent linearly separable function (image from [23]).

3 Data Acquisition

In this chapter we will introduce the three datasets used for experimentation. For each of them, traffic has been captured on the devices' 3G interface using `tcpdump`[1].

3.1 User Data

This dataset consists of recorded traffic from five distinct users, for whom all 3G network communication has been captured in the background for approximately eight hours. During this collection phase the users were interacting with their smartphones as usual, without any restrictions. Some more details of the traffic dumps can be found in Table 3.1.

User	Device	#Packets	#Bursts
1	iPhone 4	21609	161
2	Samsung Galaxy S2	9952	587
3	Samsung Galaxy S1	13342	289
4	Google Nexus S	4091	135
5	Samsung Galaxy S1	5716	261

Table 3.1: User traffic dump details.

3.2 Non-interactive Data

Due to the low scope of the user dataset, we emulated 20 user devices by installing different combinations of Apps on a smartphone. We call the dataset non-interactive because transmissions were captured in an unused mode, without any user interactions that could cause traffic.
Our testbed was a Samsung Galaxy Nexus running the Android operating system version 4.0.4. In terms of data gathering, we randomly picked 20 distinct combinations out of a universe of 14 Apps. Each combination is composed of seven Apps, whereas we assured marginal cases to be present, meaning two combinations with

[1]`http://www.tcpdump.org/`

six common applications and only one differing, as well as two combinations with completely disjoint subsets.
The 14 Apps were selected from the list of top free Android applications from the Google Play Store. Certainly, we only picked Apps that actually produce background transmissions without user interactions, since otherwise they would have no influence on the device's traffic behaviour. This comprises cloud services, several messengers or news feeds. The complete list of chosen Apps as well as some information on them can be found in Table 3.2.
An exception is list item 15, the Google account, which backs up and synchronises the phone state like browser settings, calendar events, contacts, Email and photos. The traffic of the Google account is inherently present in each of our combinations and brings further individualisation opportunities.
Analogous to the user dataset, packets were captured for eight hours. The compositions of the single measurements as well as further details are listed in Table 3.3. The App column indicates the set of indices from Table 3.2 that are part of this combination. The indices of the marginal cases are marked with special symbols.

Index	Name	Downloads[2]	Rank[3]
1	Email	-	native
2	Facebook	100 - 500	4
3	WhatsApp	50 - 100	9
4	Skype	50 - 100	12
5	Twitter	50 - 100	14
6	Dropbox	10 - 50	15
7	Instagram	10 - 50	23
8	Flipboard	5 - 10	27
9	Viber	10 - 50	34
10	Evernote	10 - 50	156
11	Spotify	10 - 50	66
12	Wetter.com	5 - 10	not ranked
13	Skydrive	0.1 - 0.5	422
14	ChatON	10 - 50	50
15	Google Account	-	native

Table 3.2: Universe of applications from which the emulating combinations were chosen.

Index	Apps	#Packets	#Bursts
*1	1,5,6,7,10,12,13	7412	247
†2	1,3,4,10,11,13,14	32623	894
3	1,3,6,8,12,13,14	9669	359
4	3,5,6,8,9,12,13	4796	487
5	2,3,5,7,8,9,14	7061	510
†6	1,3,4,6,10,11,13	29211	901
7	1,2,3,5,6,8,12	10403	333
8	1,2,6,9,11,12,13	11104	804
9	1,6,8,9,10,13,14	4139	341
10	1,2,6,7,10,11,13	21043	776
11	1,4,7,8,10,12,14	38128	371
12	2,4,6,9,10,13,14	23460	641
*13	2,3,4,8,9,11,14	57840	1120
14	1,3,5,7,9,11,13	13371	942
15	2,4,5,7,10,13,14	45685	320
16	1,4,5,9,10,12,13	18088	532
17	2,3,5,8,11,13,14	13829	853
‡18	1,2,4,7,9,10,11	32581	942
‡19	1,2,4,7,9,10,12	26031	608
20	1,5,6,9,10,12,14	5306	530

Table 3.3: Details on the single non-interactive App combinations. Row indices marked with * denote the marginal case with disjoint subsets, † and ‡ show combinations which have all but one App in common.

3.3 Interactive Data

In addition to the non-interactive dataset, which represents the unused mode of devices, we are also interested in capturing traffic that contains transmissions caused by user interactions. This makes the subsequent experiments on identification more realistic because we are able to capture both modes, interactive as well as non-interactive. Since background traffic is also present in interaction mode, we base our newly created dataset on the latter by simply injecting random interaction patterns. To achieve this, we recorded traffic evoking user interactions for the Apps from Table 3.2 as well as several other applications like Google Maps, Browser and YouTube. An interaction could be for example surfing on a webpage, logging into Skype, posting a tweet on Twitter or similar things, whereas each interaction itself may consist of one or more bursts. Now for each combination of the non-interactive dataset, we choose n random interactions and superimpose these burst patterns at random positions in

[2]Numbers $\cdot 10^6$ from `https://play.google.com/store/apps`, effective 26/09/2012

[3]Numbers from `https://play.google.com/store/apps`, effective 26/09/2012

the timeline of the non-interactive traffic dumps. We only inject interactions from Apps that are actually installed in a particular combination as well as the additional ones like Browser, YouTube etc. that every user tends to have installed. Figure 6.5 illustrates the injection of random burst patterns into the non-interactive recordings. Red patterns symbolise the interactions which might be composed by several bursts. The black shapes are parts of the non-interactive dataset and exist regardless whether there are user interactions or not.

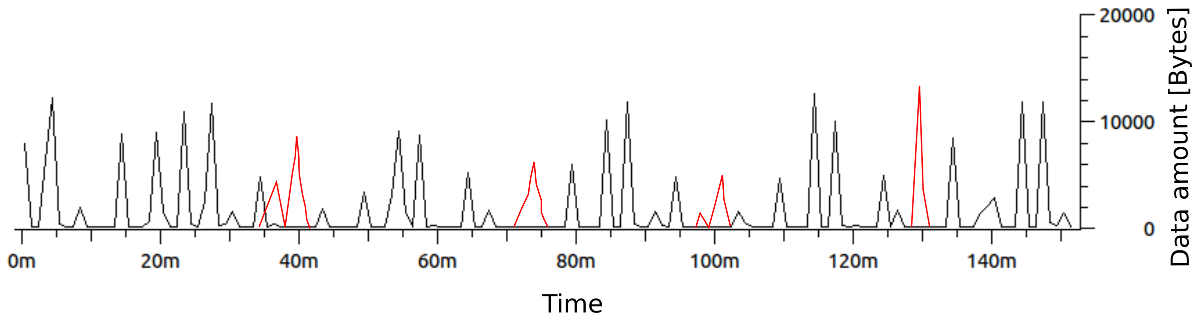

Figure 3.1: Illustration of injecting random interactions into a non-interactive traffic dump.

4 Fingerprinting

4.1 Process

The following sections of this chapter will give a detailed description of the fingerprinting process in all its stages as depicted in Figure 4.1. Subsequently, a notion of fingerprint quality will be introduced.
As input, the process requires traffic extracted from a specific device. The traffic is a chronological sequence of incoming and outgoing packets, whereas each packet is represented as a vector $p_i = (t_i, s_i, d_i)$. Hence, the only information needed about a packet is its arrival time t_i in the form of an absolute or relative time stamp, its size in bytes s_i and the direction d_i in terms of a Boolean flag distinguishing between incoming and outgoing packets.

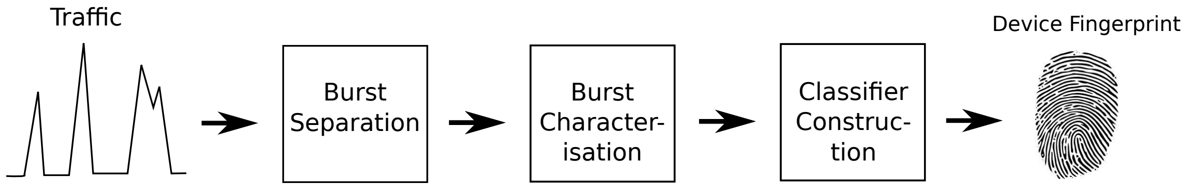

Figure 4.1: Process of generating smartphone fingerprints from captured raw traffic.

4.2 Burst Separation

In a first step, the bursts have to be extracted from the raw traffic dump. Since we cannot analyse the payload to identify and aggregate packets from the same application (using e.g. TCP flows), the bursts are created by only considering the timing information. We define a *burst distance* to be the minimum length of an idle interval between two subsequent packet arrivals in order to be considered as belonging to different bursts. Hence, extracting bursts from the captured traffic is equivalent to setting cuts in the packet sequence and aggregating all packets which are within these borders to one burst. Algorithm 1 sketches the separation process.

Since our overall goal is to classify user devices by means of their bursts, it is desirable to have reappearing occurrences of bursts with similar or even identical characteristics. This results in a small intra-class variance, which in turn facilitates the decision

Algorithm 1 Divide the ordered packet sequence *packets* into bursts.

```
define $bursts = [\,]$
define $new\_burst = [\,]$

for all $p_i \in packets$ do
    $new\_burst.append(p_i)$
    if $t_{i+1} - t_i > burst\_distance$ then
        $bursts.append(new\_burst)$
        $new\_burst = [\,]$
    end if
end for
```

of the classifier. To obtain these repetitive patterns, we approximate the burst distance such that bursts tend to consist of single transmissions. Naturally, this is not 100% achievable since transmissions from different Apps/processes/threads can occur concurrently in an interleaved manner.
Thus, selecting the burst distance is a crucial parameter. On the one hand, we may prefer a small distance because this would avoid the aggregation of several unrelated transmissions. On the other hand, choosing a too small distance will result in splitting a single transmission at points where the inter-packet delay is longer than usual, due to longer processing times or higher network congestion. Figure 4.2 depicts the number of extracted bursts as a function of the burst distance among the different datasets. An appropriate setting of the parameter would lie at some point shortly after the curve is bending. Separation before the bend is likely to result in the loss of of semantically connected packets.

The packet interarrival time is another discriminative feature. This can be seen as follows. In Figure 4.3, 90% of all packets arrive at most 1.3 seconds and 95% of all packets arrive at most 4.43 seconds after their predecessors. As a consequence, one can assume that packets belonging to the same transmission have an inter-packet delay of approximately 4.5 seconds at maximum. In [11], Falaki et al. conducted a similar experiment on a bigger set of smartphones and observed that 95% of the packets experience an interarrival time of at most 4.5 seconds. Our results confirm this outcome and the burst distance value of 4.5 seconds turned out to be a good parameter setting for the burst separation procedure.

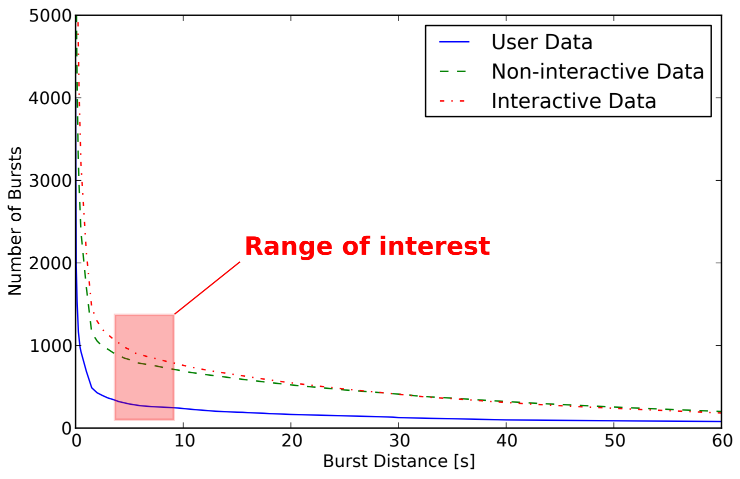

Figure 4.2: Number of extracted bursts depending on the burst distance. In order to avoid splitting two semantically connected packets and to avoid aggregating multiple unrelated transmission in one burst, a good value must be chosen from the range of interest.

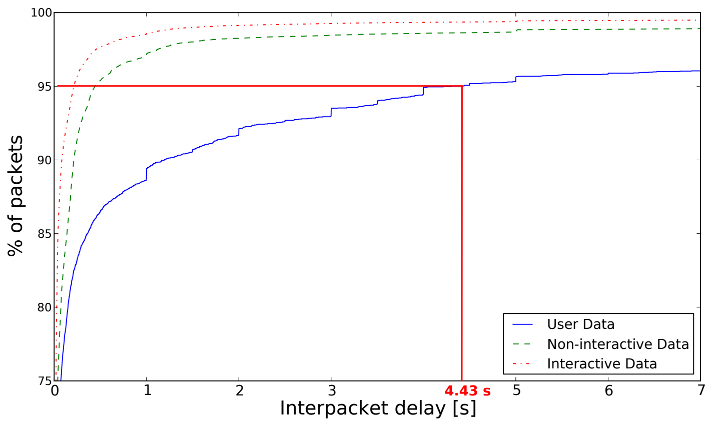

Figure 4.3: Empirical CDF of packet interarrival times. 95% of all packets are received at least 4.43 seconds after their predecessors.

4.3 Burst Characterisation

In order to be later on able to compare two bursts, we have to characterise them by means of some attributes, also known as *features*. In the next step, we identify the most discriminative features that distinguish well between bursts generated by different smartphones when these bursts are projected on the selected feature space.

4.3.1 Feature Selection

Feature selection is essentially a technique of selecting the most relevant and discriminative subset from the overall feature set. In our case, the purpose of this analysis is not primarily to minimise computational costs in the training phase or to address the curse of dimensionality. We are rather interested in gaining a better understanding of this particular kind of data.

As an illustrative example for feature selection, imagine a set of people out of which you want to classify whether a single person is a doctor or not, just by a given number of characteristics. Therefore consider Table 4.1, whereas each line corresponds to an observation and each column denotes a certain feature. By contrast, the last column shows the data labelling, indicating whether a person is a doctor (1) or not (0). Thus, this is a question of supervised learning.

Feature / Person	Salary	Height	Period of Education	Label
1	80.000	182	18	1
2	60.000	190	17	1
3	90.000	170	20	1
4	85.000	195	10	0
5	40.000	184	9	0
6	35.000	169	13	0

Table 4.1: Example of a labelled observation set.

It is trivial to see and understand that in order to distinguish between doctors and non-doctors, the features *Salary* and *Period of Education* are very helpful. In contrast to that, the *Height* of people is uncorrelated to the grouping. Hence, we would call the former features discriminative in this case, as they help to differentiate between persons of the two classes.

Our aim is to detect the most important and discriminative variables and find out which of them are potentially redundant or related to each other. In Section 4.3.1 we compute the mutual information to achieve a ranking based on the feature informativeness. Afterwards, we will calculate the correlation of single variables with each other in Section 4.3.1.

In terms of burst description, we propose a list of 24 features that can be found in Table 4.2. Since our fingerprinting and identification method solely relies on link layer information, the variables only range in two dimensions: the time domain and the data amount domain.
In addition to the mean values of the packet interarrival times and the packet sizes, we also take the 20%, 50% (median) and 80% quantiles of the timing and size distributions into account. In the presence of outliers and non-Gaussian distributed feature

values, these measures are more robust in comparison to the simple arithmetic mean. For the same reason, we apply the median absolute deviation (MAD) in terms of packet sizes and packet interarrival times. The MAD is the median of the deviations from the median [16]. Let X be a sample set vector, the medium absolute deviation is computed as

$$MAD = median(|X - median(X)|). \tag{4.1}$$

Moreover, in order to offer a point of reference, we introduced an artificial feature with random values that should hardly give any information about the user.

Feature Importance

Feature	**rMI**
Median absolute deviation (MAD) packet size	16.6%
50% quantile packet size	15.7%
Standard deviation packet size	15.6%
80% quantile packet size	14.7%
20% quantile packet size	14.6%
Mean packet size	13.8%
Byte ratio	13.8%
Distance to next burst	8.5%
Number of outgoing bytes	7.2%
80% quantile packet interarrival time	7.0%
Mean packet interarrival time	6.9%
Number outgoing packets	6.9%
Packet ratio	6.5%
50% quantile packet interarrival time	6.2%
Throughput	5.9%
Duration	5.8%
Number of incoming packets	5.7%
Number of incoming bytes	5.6%
Median absolute deviation packet interarrival time	5.5%
Standard deviation packet interarrival time	5.5%
Mean consecutive outgoing packets	4.8%
20% quantile packet interarrival time	4.5%
Mean consecutive incoming packets	4.2%
Random feature	0.9%

Table 4.2: Feature list with respective relative mutual information (rMI) for the non-interactive dataset.

To determine the importance of single features, we compute the mutual information (MI) between the features and the target variable (here the user ID). We use MI as an information-theoretic measure that indicates the informativeness of a particular random variable with respect to another random variable. In our application, it quantifies how much the knowledge of a certain feature F_i contributes and helps to correctly assign an observation to its corresponding class U.

$$I(F_i; U) = H(U) - H(U|F_i) \tag{4.2}$$

$$\mathrm{rMI}(F_i; U) = \frac{I(F_i; U)}{H(U)} \tag{4.3}$$

Equation (4.2) defines the mutual information in terms of entropy, whereas F_i and U represent a feature i and the user IDs, respectively. The entropy $H(U)$ quantifies the uncertainty about the value of U. The conditional entropy $H(U|F_i)$ quantifies the remaining uncertainty if the value of feature i is known. The difference of $H(U)$ and $H(U|F_i)$ becomes maximal if the feature fully determines the user.

Before computing MI, it is necessary to quantise the feature values if they are continuous. Therefore, we divide the feature co-domain into a fixed amount of intervals, so-called bins, and map every value within a certain interval to the bin's number. Furthermore, in order to avoid a too wide range caused by outliers, it is useful to consider the co-domain only in a subinterval. In our case, taking a look at the distributions of single variable values revealed that there are almost exclusively outliers on the right side of the spectrum. Thus, we narrowed the co-domain from the 0%- to the 90%-quantile in our computations, accounting all outliers to the last quantile.

Figure 4.4 shows the relative mutual information of the features for different combinations of quantile values. For both plots (non-interactive data and user data), the black bars, representing the 0%- and 90%-quantile subrange, always reach the largest rMI for all features. As stated above, this can be explained by the presence of outliers especially on the right side of the respective co-domain.

One can observe that the distribution of importance among the features is nearly the same for both datasets. However, the features have a higher absolute informativeness in the user dataset case which can be explained by the relatively small sample size. As expected, the random feature barely carries any information about the user.

As a result of our analysis, we refrain from choosing only the best ranked features. The reason is that even variables with a small independent informativeness can provide rich information when combined. This is exactly the case if the two variables describe complementary parts of the classification target [15]. In contrast, pairs of variables with large rMI could be fully redundant.

Feature Correlation

The objective of this section is to get an insight about which features provide redundant information and could possibly be omitted in further computations. The degree of redundancy can be illustrated in terms of correlation.

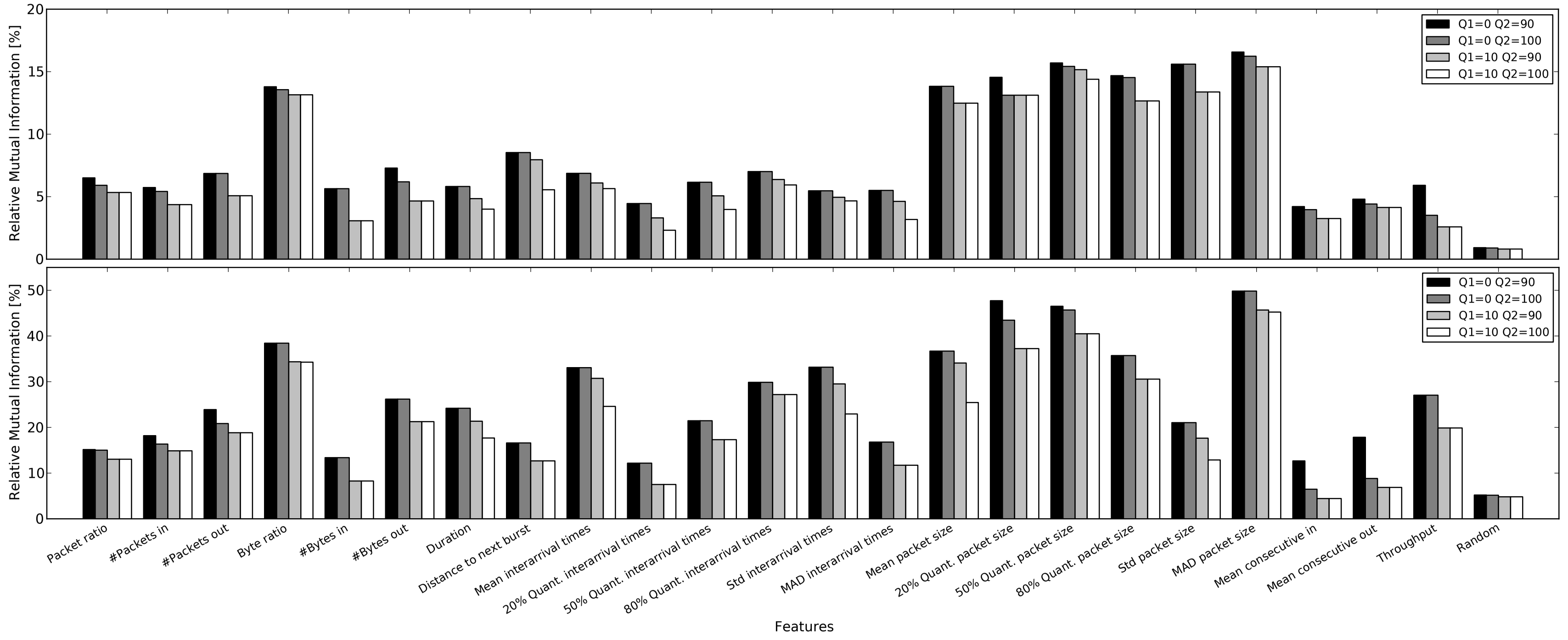

Figure 4.4: Relative mutual information of features concerning different quantile combinations for their co-domains. The upper chart is based on the non-interactive, the lower one on the user dataset.

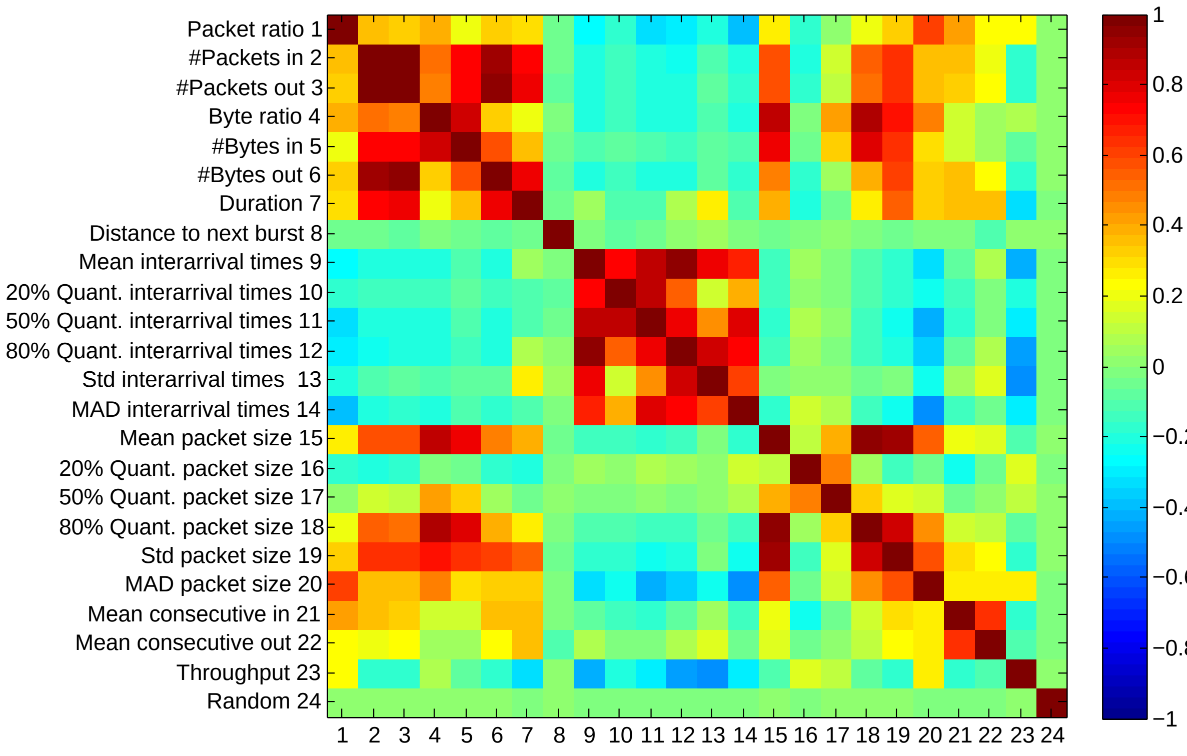

Figure 4.5: Feature correlation matrix of Pearson correlation coefficients.

We compute the Pearson correlation coefficients for all pairwise feature combinations as follows. Let X be the observation matrix, whereas X_i corresponds to the i^{th} column vector and contains all values of the i^{th} feature, the coefficients $R(i, j)$ can be calculated as

$$R(i,j) = \frac{cov(X_i, X_j)}{\sqrt{var(X_i) \cdot var(X_j)}}. \tag{4.4}$$

The outcome can be inspected in the colour matrix of Figure 4.5. The largest positive correlation is computed for the number of incoming- and outgoing bytes and packets. Quite intuitive is also the correlation between the amount of received/transmitted packets and the burst duration, since the more data has to be transmitted the longer it takes. Finally, again as expected, the random feature is uncorrelated to every other variable. A very distinct feature is the distance to the next burst which is, just like the random feature, uncorrelated to every other variable but, yet it carries information about the user ID.

According to [15], perfectly correlated variables are redundant in the sense that no additional information is gained when adding them. However, high feature correlation or anti-correlation can still mean that features complement each other. Hence, even the usage of two strongly correlated features may still improve the process of classification.

4.4 Classifier Construction

In this last step, we have to construct the actual fingerprint. Certainly, the sequence of extracted bursts can already be seen as a fingerprint. However, in order to be able to accomplish the matching procedure, we train a classifier for each phone and consider it to be the fingerprint of the device.

As it is common practice in supervised learning, we turn this multi-class classification problem with n user IDs into n individual binary classification problems. In each individual problem, a classifier must decide if the current data comes from the respective user or not. For each such problem, half of the observation matrix, used as an input for training the classifier, consists of bursts from the legit user. The remaining 50% are filled by an equal number of bursts from other users. This way, we only have two class labels, namely *user* and $\neg$*user*. Each observation or burst in the observation matrix is represented by a row whereas each of the 23 columns corresponds to a feature. As introduced in the foundation chapter, we utilise an SVM as well as the *k*NN algorithm. To estimate the optimal setting for the parameter *k*, we perform a five-fold cross-validation on the training data in advance. The value for *k* is chosen from the set of all odd numbers from 1 to 15. A small snippet of Matlab code responsible for constructing the *k*NN classifier can be found in Listing 4.1.

```matlab
% contruct classifier and tune parameter k by cross-validation
mdl = ClassificationKNN.fit(x_train, y_train);
numNeighbors = [1 3 5 7 9 11 13 15];
minK = 1;
minKloss = 1.0;
for k=numNeighbors;
    mdl.NumNeighbors = k;
    % Construct a cross-validated classifier from the model
    cvmdl = crossval(mdl, 'kfold', 5);
    kloss = kfoldLoss(cvmdl);
    if kloss < minKloss;
        minKloss = kloss;
        minK = k;
    end
end
% choose k with the minimal loss
mdl.NumNeighbors = minK;
```

Listing 4.1: "Training" the *k*NN classifier with cross-validation

4.5 Fingerprint Quality

Considering a smartphone fingerprint, quality can basically be split up into two measures, **significance** and **robutstness**.
Significance describes a fingerprint to be expressive and convincing, which essentially relates to the quantity of contributing bursts as well as Apps. This measure is also related to **uniqueness**, since it is more likely to encounter a similar fingerprint if only a small set of Apps contributes to it. Robustness states how susceptible a fingerprint is to certain changes in traffic patterns. Both measures are subject to several factors.

1. **Number of contributing Apps**
   This measure counts the number of Apps that generate the traffic and hence contribute to the fingerprint. The larger this number, the less likely it is to find another device with exactly the same configuration and combination of installed Apps. Thus, the fingerprint becomes more significant and, since uninstalling one out of many Apps only slightly changes the phone state compared to uninstalling one out of a set of three Apps, the fingerprint also gets more robust to configuration changes. An insufficient number of contributing applications results in a very general, less discriminative device state, impairing significance as well as robustness.
2. **Type of contribution**
   In addition to the quality of an entire fingerprint, we can also define the discriminative quality of an individual App. If an application generates bursts in a random manner such that the probability for two bursts being similar is low, it is not useful for the classification problem. This can be understood considering the k-nearest neighbour algorithm which looks for the k most similar burst occurrences and accordingly assigns class labels. Therefore, we can approximate the quality of a single application by the degree of mutual similarity between bursts arising from the same App.
3. **Burst distribution**
   Suppose a distribution where 90% of the bursts arise from one individually configured application and 10% arise from several other Apps. In terms of robustness, uninstalling this single App would result in a drastic change of the fingerprint, what exacerbates the subsequent identification.
   Hence, in the best case, the bursts are uniformly distributed across the set of Apps.
4. **Quantity**
   Especially for the significance, the overall number of bursts belonging to the fingerprint plays an important part. Imagine a fingerprint consisting of ten high quality Apps with uniformly distributed bursts, but only 20 bursts all in all. Even though the other factors are nearly optimal, the fingerprint is not sound due to the small sample set.

4.5.1 Derivation of a Quality Approximation

Definition 1. *Let f be a fingerprint consisting of a sequence of bursts B_f. We define an App set $A_f = (a_1, \ldots, a_n)$ that is composed of all Apps contributing to the device's traffic. Every $a_i = (b_{i1}, \ldots, b_{im_i}) \in A_f$ is a subset of B_f with the property that $\sum_{i=1}^{n} |a_i| \approx |B_f|$.*

The latter equation holds only approximately due to the fact that a burst may consist of several interleaved transmissions from distinct Apps.

Before we come to the definition of the fingerprint quality, we have to estimate the quality of a single application (Q_a). As already mentioned, we declare the quality of an App by means of similarity between the contained bursts. To be robust against outliers, we pick the median of all mutual burst similarities and define the App quality as

$$Q_a = s_a = median(\{s_{b_{ij}} | \; \forall i \neq j \in \{1, \ldots, |a|\}\}). \tag{4.5}$$

The similarity $s_{b_{ij}}$ between two bursts can be computed by some distance metric (e.g. Euclidean distance) $dist(b_i, b_j)$ of the feature vectors. Because of the relation of the similarity function *sim* and the distance function *dist*, we can easily derive the similarity of two bursts on the basis of the distance (see Equation 4.6).

$$sim(b_i, b_j) = 1 - dist(b_i, b_j) \tag{4.6}$$

In the sequel, we will derive a formula for the fingerprint quality (Q_f) step-by-step. A first proposal would be to simply compute the mean of all single App qualities (cf. Equation 4.7). However, an easy counter example to demonstrate that this approach does not fulfil our requirements is the scenario explained in point 3 of Enumeration 4.5. Accordingly, having one App to which 90% of all bursts belong and several others sharing the residual 10%, the uninstalling of one App could cause a significant change of the fingerprint, even if all single App qualities were high. In other words, the lacking robustness is not expressed in the quality measure of Equation 4.7.

$$Q_f = mean(\{Q_a | \; \forall a \in A_f\}) \tag{4.7}$$

In order to fix this, we need to include a measure that reaches its maximum in case of a uniform distribution. This is where the entropy comes in (cf. Equation 4.8). For the probabilities we simply take the burst frequencies of the single Apps divided by the overall number of bursts. The entropy of a fingerprint H_f will maximise if and only if the bursts are uniformly distributed across all applications. We will refer to this maximum entropy as H_{max}.

$$H_f = -\sum_{i=1}^{n} \frac{|a_i|}{|B_f|} \cdot log_2\left(\frac{|a_i|}{|B_f|}\right) \tag{4.8}$$

Enhanced by the distribution measure, the second suggestion for the fingerprint quality can be found in Equation 4.9. However, this approach is not reliable either, because it neither considers the size of the sample set B_f nor the amount of contributing Apps. Applying Equation 4.9, a fingerprint composed by 20 bursts and 2 contributing Apps can reach the same quality as a fingerprint consisting of 500 bursts arisen from 15 Apps.

$$Q_f = mean\left(mean(\{s_a|\forall a \in A_f\}),\ \frac{H_f}{H_{max}}\right) \tag{4.9}$$

Hence, in a last step we have to take into account the quantities $|B_f|$ and $|A_f|$. Since we want to end up with a quality measure in the interval of $[0,1]$, $|B_f|$ and $|A_f|$ need to be relativised. Therefore, we define thresholds B_{max} and A_{max} which state that values above these limits do not bring significant advantage anymore. For instance a fingerprint comprised of 1000 bursts is not significantly more meaningful than one consisting of 500. All in all, we define the quality of a fingerprint in Equation 4.10. The numbers beneath the brackets represent the factor indices from Enumeration 4.5.

$$Q_f = mean\left(\underbrace{mean(\{s_a|\forall a \in A_f\})}_{2},\ \underbrace{\frac{H_f}{H_{max}}}_{3},\ \underbrace{min\left(\frac{|A_f|}{A_{max}}\right)}_{1},\ \underbrace{min\left(\frac{|B_f|}{B_{max}}\right)}_{4}\right) \tag{4.10}$$

Table 4.3 shows the estimated fingerprint qualities for all combinations of Apps from our non-interactive dataset. The column Q_a indicates the degree of intra-application similarity of bursts. H_f quantifies how uniformly the bursts are distributed across the applications. One can for instance observe that especially the combinations including the Spotify App tend to have a poor quality factor in terms of their burst distribution due to its high traffic domination.

Clearly, the quality of a single fingerprint is not the only crucial factor for an authentification or identification problem. Even the best password becomes futile if everybody uses nearly the same. Hence, it is at least as important to regard the overall set of smartphone fingerprints respectively traffic behaviours and obtain an insight about their discriminability.
Roughly speaking, this relates to the question of how probable it is for several devices to share the same combination of installed applications.

Index	Q_a	$\frac{H_f}{H_{max}}$	Q_f
1	0.64	0.80	0.82
2	0.60	0.52	0.78
3	0.63	0.92	0.89
4	0.61	0.87	0.87
5	0.51	0.55	0.77
6	0.53	0.50	0.76
7	0.73	0.89	0.90
8	0.58	0.41	0.75
9	0.64	0.66	0.83
10	0.48	0.32	0.70
11	0.56	0.96	0.88
12	0.74	0.59	0.83
13	0.59	0.72	0.83
14	0.51	0.36	0.72
15	0.72	0.93	0.91
16	0.66	0.63	0.82
17	0.57	0.44	0.75
18	0.65	0.38	0.76
19	0.59	0.68	0.82
20	0.56	0.47	0.76

Table 4.3: Estimated fingerprint qualities for the 20 App combinations of the non-interactive dataset. The indices relate to the combination numbers of Table 3.3.

5 Identification

5.1 Process

Analog to the fingerprint generation, the process of identifying a smartphone user, as depicted in Figure 5.1, will be introduced in the following sections.
In the first place, traffic has to be captured. The identification can either happen online in real-time or delayed by means of an already recorded traffic dump. In the latter case, an adversary could make a point of whether the smartphone owner was at a certain place at a certain time in the past. Note that the trivial prerequisite for the whole identification procedure is the availability of the victim's fingerprint.

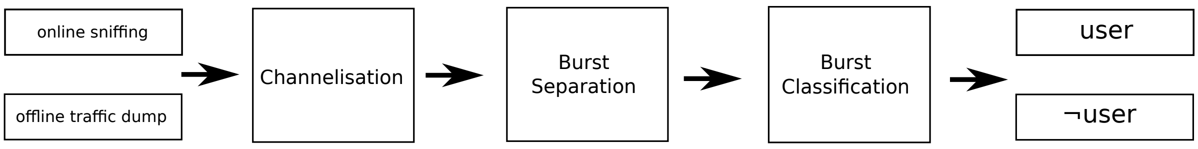

Figure 5.1: Process of identifying a smartphone user based on its fingerprint.

5.2 Channelisation

Apart from the just mentioned prerequisite of being in possession of a fingerprint, the adversary must also be able to channelise the captured traffic, independent on which wireless communication technology the attack is based on. Channelisation is the process of demultiplexing the potentially multiplexed radio signal in order to obtain separated user datastreams.
In the UMTS scenario, this implies the knowledge or "brute-forcing" of scrambling and spreading codes, whereas in e.g. an IEEE 802.11 network, it is simply achievable by MAC address separation. However, channelisation is not to be confused with the requirement of knowing the actual mapping between a particular user datastream and the corresponding user.

5.3 Burst Separation

The burst separation procedure is equivalent to the one in Section 4.2. By simply considering interarrival times, the packet sequence is divided into bursts. Importantly, the setting for the burst distance parameter ought to be the same as in the fingerprinting process.

5.4 Burst Classification

After extracting the bursts from a user datastream, they are, either individually or combined as a vector, fed into the trained classifier.
Classifying n bursts will result in an outcome of n labels, saying either *user* or $\neg$*user*. To obtain a final decision, one can simply take the majority vote of labels from the result vector or determine some threshold that has to be exceeded in order to consider the traffic as a fingerprint match.
Listings 5.1 and 5.2 show the corresponding Matlab code for the actual classification procedure with the *k*NN algorithm and the SVM, respectively. The captured bursts are represented as an observation vector `x_test`, whereas the fingerprints are represented by the constructed classifiers, indicated in the first line of both listings. The instruction `mode(res)` simply outputs the label with the most occurrences in the result vector `res`.

```
mdl = ClassificationKNN.fit(x_train, y_train);
res = mdl.predict(x_test);
mode(res)
```

Listing 5.1: *k*NN classification of captured bursts.

```
svmstruct = svmtrain(x_train, y_train, 'kernel_function', 'rbf');
res = svmclassify(svmstruct, x_test);
mode(res)
```

Listing 5.2: SVM classification of captured bursts.

6 Experimental Evaluation

6.1 Single Burst Classification

6.1.1 Objective

The goal of this experiment is to investigate the feasibility of our fingerprinting and identification approach. On the one hand, this requires to demonstrate that traffic from a particular user u matches its own fingerprint such that the majority of bursts gest classified correctly. On the other hand, we have to check that no other traffic matches u's fingerprint and also that u's traffic does not apply to any other fingerprint.

6.1.2 Design

Talking about a cartain user from both the non-interactive and the interactive dataset, we always refer to a particular App combination, as these emulate different users. For each user u out of the dataset, a fingerprint was constructed by training a classifier with 70% of u's bursts and the same amount of bursts from other users, both randomly chosen. Due to the fact that there are only two possibilities to assign a burst, either to class u or to class $\neg u$, this results in a binary classification. As stated in Section 4.4, in each case we applied the kNN classifier and an SVM.
After the fingerprints were generated, we started classifying the remaining 30% of u's bursts as well as evenly many random bursts from every single other user, assuring not to employ any burst that has been used for building the classifier. Each classification combination was repeated in 20 rounds, in each of which the training- and test set were populated by newly and randomly selected bursts.
As a result, we obtain the number of false negative- (FN) and false positive (FP) class assignments. FNs for the cases where we match users on their own fingerprints and FPs for the remaining cases.

6.1.3 Results

Regarding the colour matrices, the line numbers denote the users' indices for whom a fingerprint was generated. According to this, cell $(13, 5)$ depicts the classification

result for matching traffic from user 5 against the fingerprint of user 13. The colours of the cells indicate how well the traffic matched the fingerprint. The brighter the colour, the more bursts have been classified to match the fingerprint. The number within the cells shows how many applications the two compared combinations have in common. The diagonal is consistently labelled with seven, since one and the same combination is matched to itself.
Figure 6.1 shows the results for the *k*NN (upper chart) and SVM (lower chart) classification of the non-interactive dataset. In general, the SVM performs better, especially in terms of the false negative rate (diagonal). The precise false positive- and false negative rates for both classifiers and both datasets are given in Table 6.1.

	User dataset		**Non-interactive dataset**	
	SVM	*k*NN	SVM	*k*NN
FNR	13.60%	16.99%	17.44%	22.85%
FPR	24.02%	31.41%	32.60%	33.46%

Table 6.1: False negative- and false positive rates for the single burst classification experiment.

The results for the user dataset (see Figure 6.2), especially when utilising an SVM classifier, are nearly optimal. Not only due to the minor scope of the dataset but also because in reality the event of two users having nearly the same combination of installed Apps with similar configurations is rather unlikely. This makes the UMTS application again attractive to attackers, since he would only have to compare the traffic to a fixed number of users per cell.
Obviously, it becomes more and more difficult for the classifier to distinguish between two users who share many installed Apps and therefore produce similar traffic. This can be seen by comparing cell $(13,1)$, which shows two users with disjoint App sets, to cell $(2,6)$ that represents the opposed marginal case of all but one shared applications. However, there are also other influences. For example the App Spotify tends to have high traffic load in terms of generating many bursts. Consequently, matching two combinations against each other that both contain Spotify will exacerbate the decision of the classifier. This can be comprehended looking at Figure 6.1. All combination indices out of the set $\{2,6,8,10,13,14,17,18\}$ that are matched with each other show significantly brighter cell colours than their neighbours. However, we utilised Spotify in each combination with the same configuration, which leads to the assumption that distinct configurations (Playlists etc.) might slightly alter the traffic and simplify the task of the classifier.

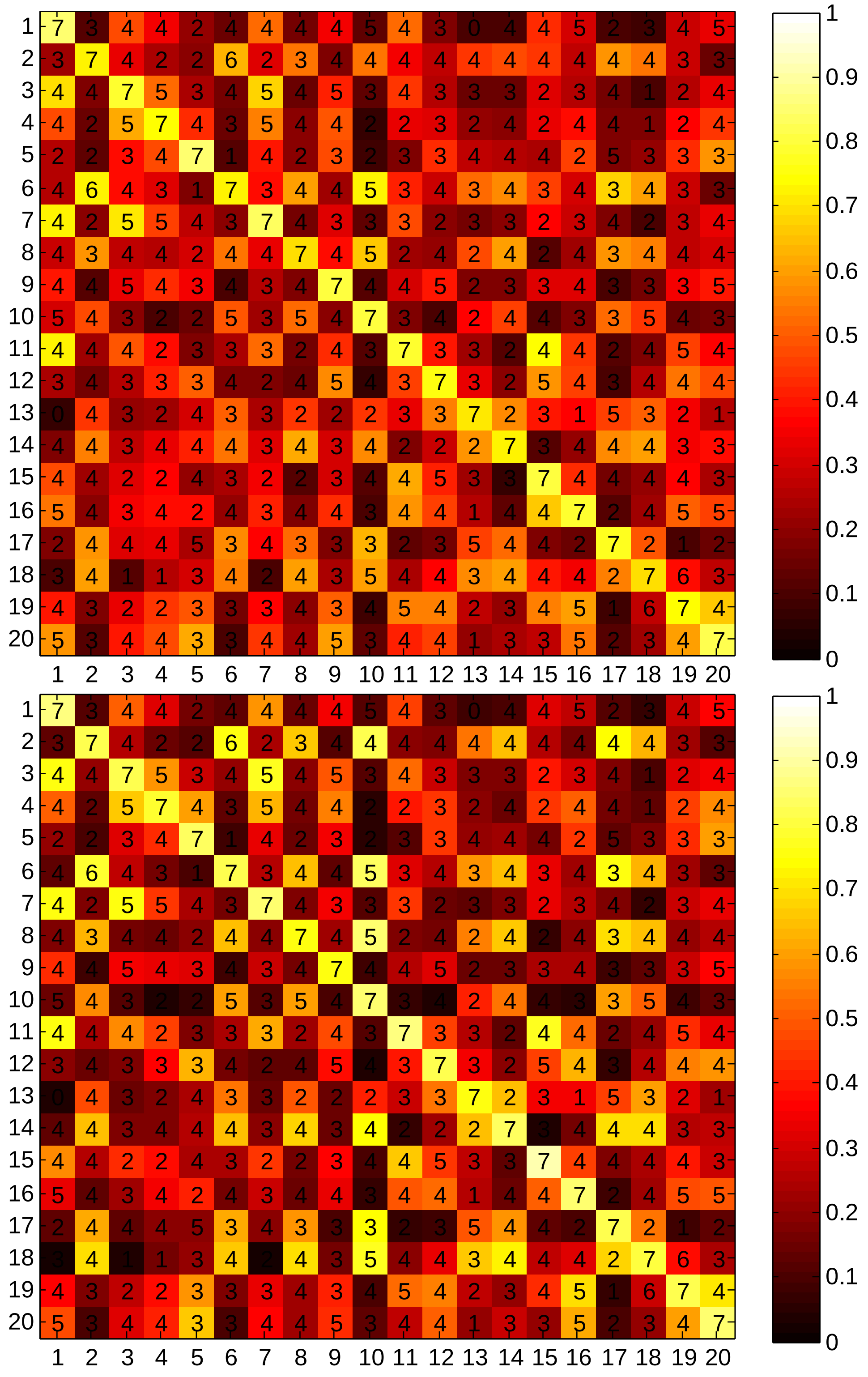

Figure 6.1: *k*NN (top) and SVM (bottom) results for matching users against every fingerprint (non-interactive dataset). The brighter the colour, the more bursts were classified to belong to the fingerprint. The diagonal represents the cases were user traffic was matched against its own fingerprint. The numbers inside the cells indicate the amount of common Apps of two combinations.

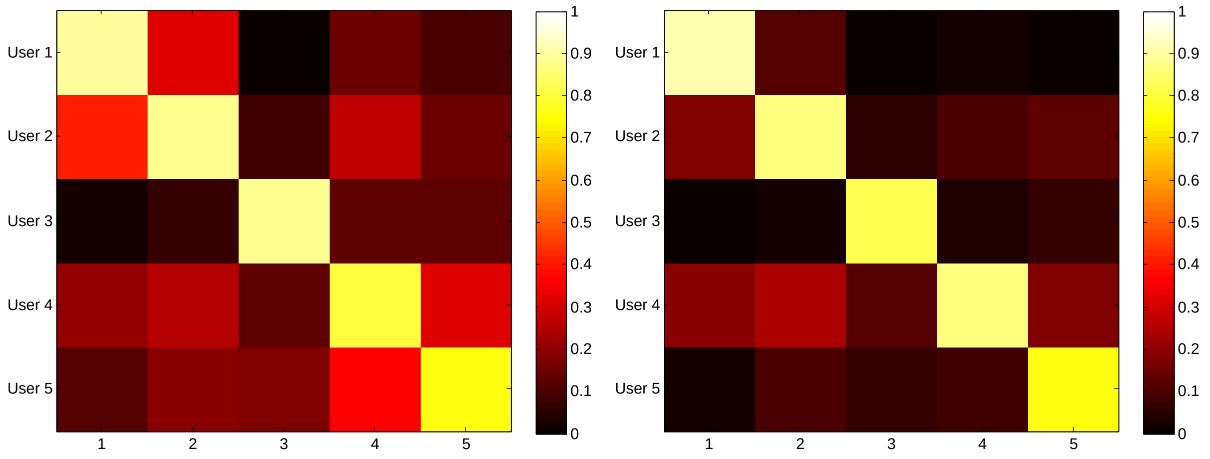

Figure 6.2: kNN (left) and SVM (right) results for matching users against every fingerprint (user dataset).

6.2 Effects of Capturing Duration

6.2.1 Objective

The objective of this experiment is to find out how long a potential attacker has to capture traffic of a smartphone in order to make a reliable statement about whether it belongs to the target user or not. On the one hand, this implies that we have to look at classification results for varying amounts of available traffic in terms of different numbers of bursts. On the other hand, we have to evaluate the burst interarrival times to find out how long an attacker has to wait approximately in order to gather a certain amount of traffic.

6.2.2 Design

As in the last experiment, we generate a fingerprint for each user with a training set containing 70% of its bursts and equally many random bursts from other users. Yet, unlike last time, the training set is filled with the residual 30% of the user's bursts *as well as* the same amount of randomly chosen bursts from other users, assuring to not employ those that have been used for training. Thus, we get false positive and false negative results.
Instead of classifying each single burst on its own, we apply a sliding window of size $k \in \{1, 2, \ldots 40\}$ such that we classify k bursts and take the majority vote of the k labels. This is similar to how an attacker would precede. Collecting a certain amount of bursts, match them against the fingerprint and take the majority vote as a final decision.

This procedure is repeated in 20 rounds for every user. The final error rate is computed by dividing the sum of false positives and false negatives by the overall amount of classifications.

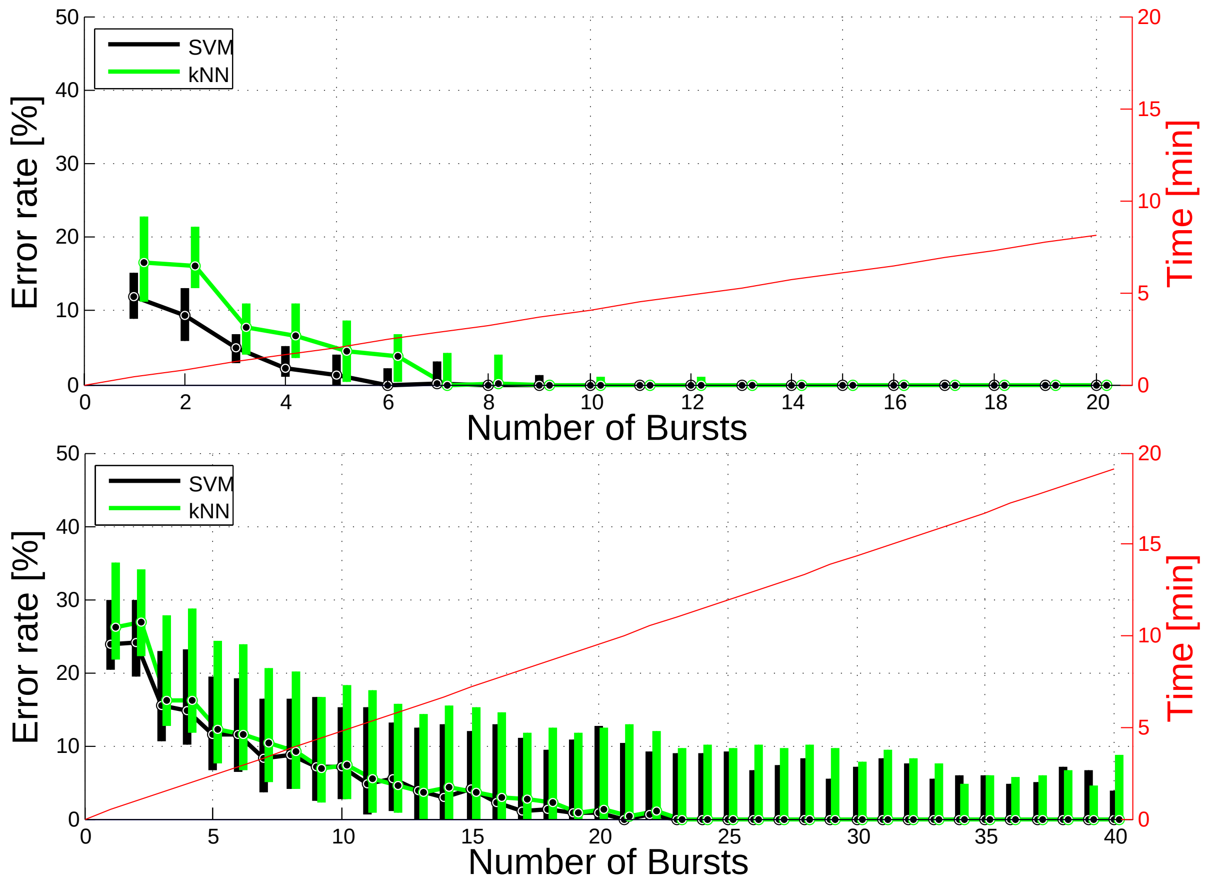

Figure 6.3: Error rate depending on the number of bursts employed for classification. To achieve a median classification error rate of 0%, an attacker would have to wait approximately 3 minutes in case of the user dataset (upper chart) and 12 minutes for the non-interactive dataset (lower chart), respectively.

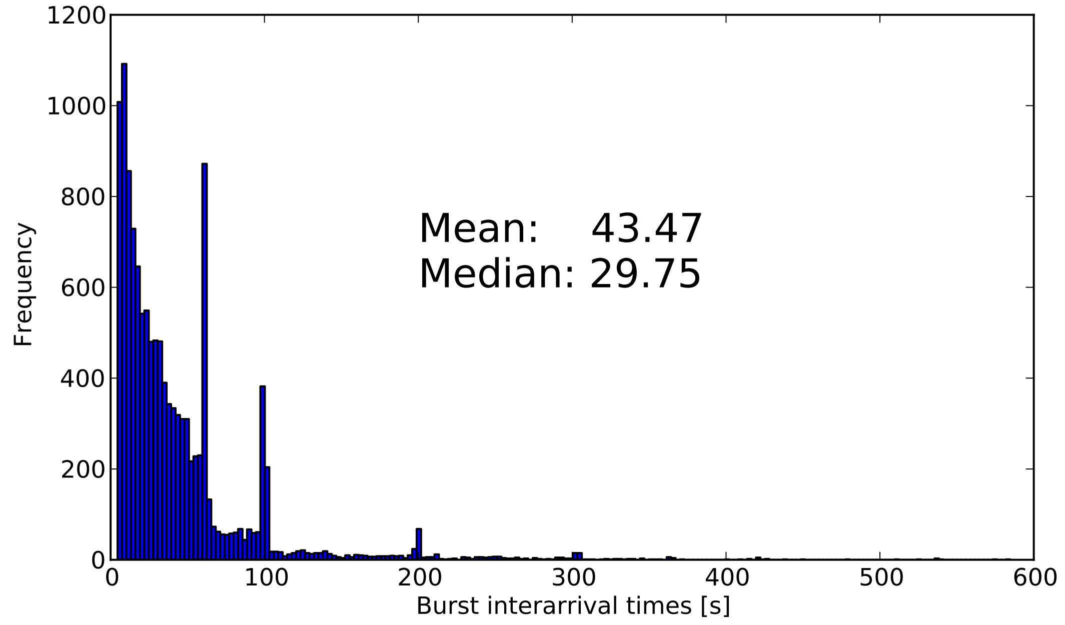

Figure 6.4: Burst interarrival times (non-interactive dataset), indicating how long an attacker approximately has to wait for a particular number of bursts.

6.2.3 Results

Both charts in Figure 6.3 show the results for the user- and the non-interactive dataset applying the *k*NN algorithm and the SVM. The bars reach from the 25% quantile to the 75% quantile whereas the dots inside the bars represent the median values. As it can be seen, the SVM generally performs better than the *k*NN classifier, such that after six bursts for the user dataset and after 23 bursts for the non-interactive dataset a median error rate of 0% is reached. According to the timeline, which is represented by the linear function, an attacker would have to wait merely ∼3 minutes and ∼12 minutes, respectively, in order to reliably identify a user. The distribution of burst interarrival times, from which the red time function is derived, can be found in Figure 6.4 for the non-interactive dataset.

6.3 Effects of User Interaction

Up to now, we only considered the user dataset as well as the non-interactive dataset. This section aims at showing the influence on the classification error rate when certain amounts of user interactions are injected. As explained in Section 3.3, the dataset was generated by injecting random user interactions in form of one or more bursts at

random points in time into the non-interactive data.
The results are depicted in the boxplot of Figure 6.5. Even with an injection of 200 interactions, which corresponds to a user causing traffic every 2.5 minutes during the whole eight hours, the median error rate only increases about 6% from 22% to 28%, which in turn indicates that our approach also succeeds for smartphones in use.

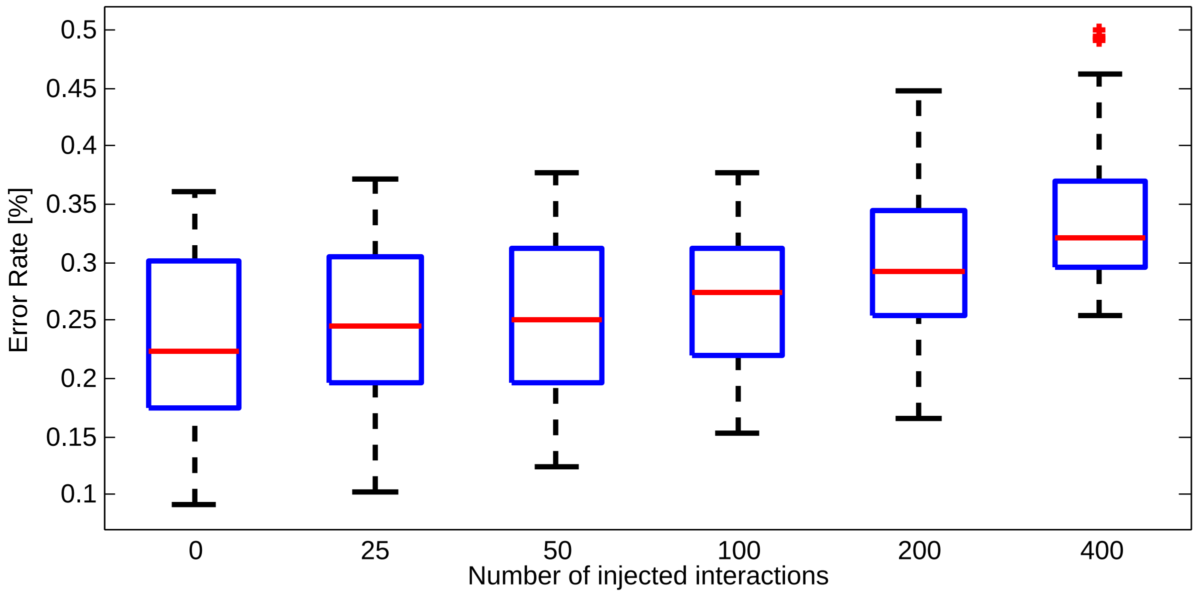

Figure 6.5: Classification error rate depending on the amount of injected interactions. Two-hundred injections relate to a user interaction every 2.5 minutes over a span of 8 hours.

6.4 Downlink Only

Especially for the UMTS use case, where it is simpler to capture traffic on the downlink from the Node-B to the user equipments, it is interesting to regard the results for only considering downlink data that was sent to the smartphones. Figure 6.6 illustrates the results of the experiment from Section 6.2 in the downlink-only case. The outcome indicates that the median error rates only impair slightly. In order to classify a user's traffic with a median error rate of 0%, the attacker would have to monitor and capture for ∼15.5 minutes compared to 12 minutes for the bidirectional case. However, in this scenario, the *k*NN classifier outperforms the SVM by far. This is most likely due to the fact that the SVM classification relies on certain features that are not available anymore in the downlink case. As can be seen in Figure 6.7, the features *Packet ratio*, *#Packets out*, *Byte ratio*, *#Bytes out* and *Mean consecutive out*, carrying uplink information, are not present anymore. Most notably, the *Byte ratio* had a comparatively high informativeness, thus a significant importance in terms of classification. In summary, it can be stated that even if only downlink data is available, the smartphone identification is still realistic.

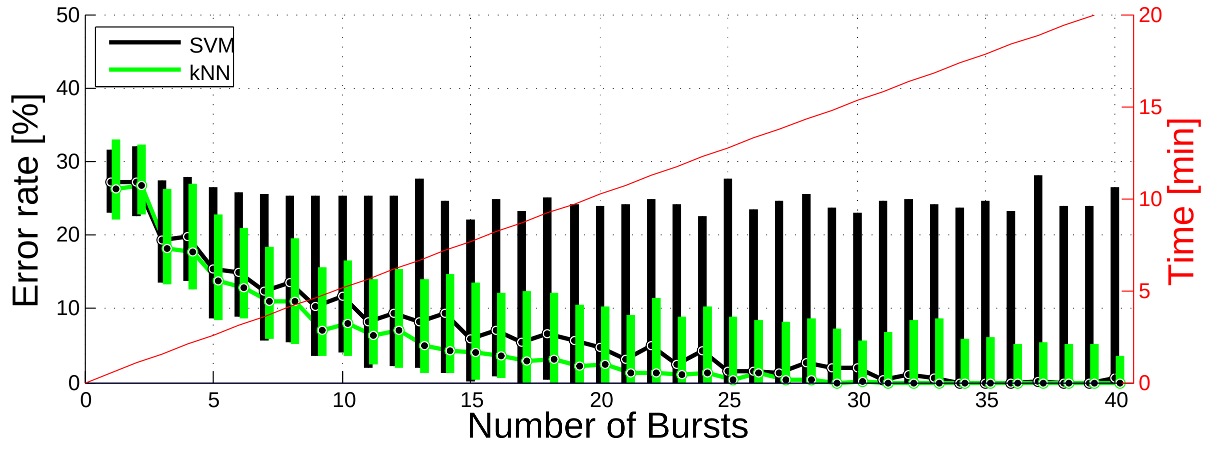

Figure 6.6: Error rate depending on the number of bursts employed for classification, taking into account only downlink data (non-interactive dataset). The linear curve gives insight on how long an attacker would have to wait for a certain number of bursts.

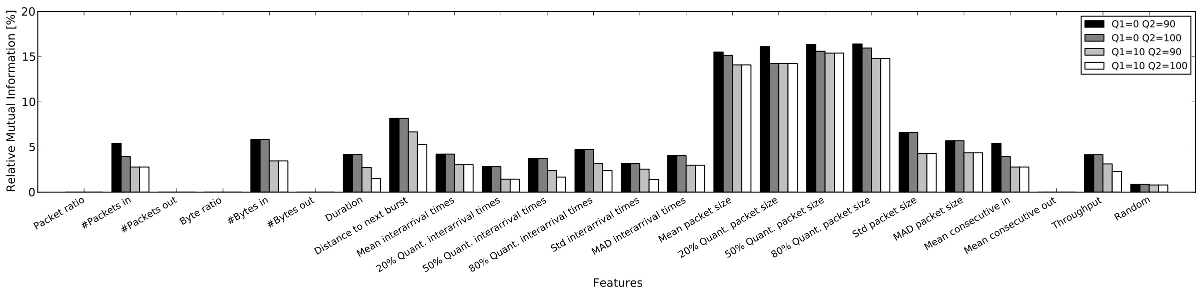

Figure 6.7: Relative mutual information of features concerning different quantile combinations for their co-domains (non-interactive dataset, downlink-only).

6.5 Why Bursts?

One of the basic concepts of this thesis is the separation of captured traffic into bursts, which are subsequently used as observations in order to construct a classifier. The goal of this experiment is to demonstrate that the traffic division in terms of bursts is actually advantageous. Therefore, we compare the results that were achieved by means of burst separation to classification results from dividing traffic into constant time slots of different lengths.
As in previous experiments, a fingerprint was built for every user by 70% of the available data and equally many bursts or constant time slots, respectively, from other

users. The residual 30% of the user's observations and again an equal amount of other observations from distinct users were classified during the test phase. For every user, this procedure was repeated in ten rounds, each of them with random separations into training and test set.
The classification results for the user dataset can be found in Figure 6.8. The y-axis shows the error rate (sum of false positive and false negative rate) for different lengths of the constant time slots as well as the original separation into bursts. All in all, the burst variant is the most effective one. However, the median error rates for the constant time interval variants are not significantly worse. Nevertheless, the bursts separation reveals much more reliable classification results, as can be seen by the small inter-quantile range and the absence of outliers.

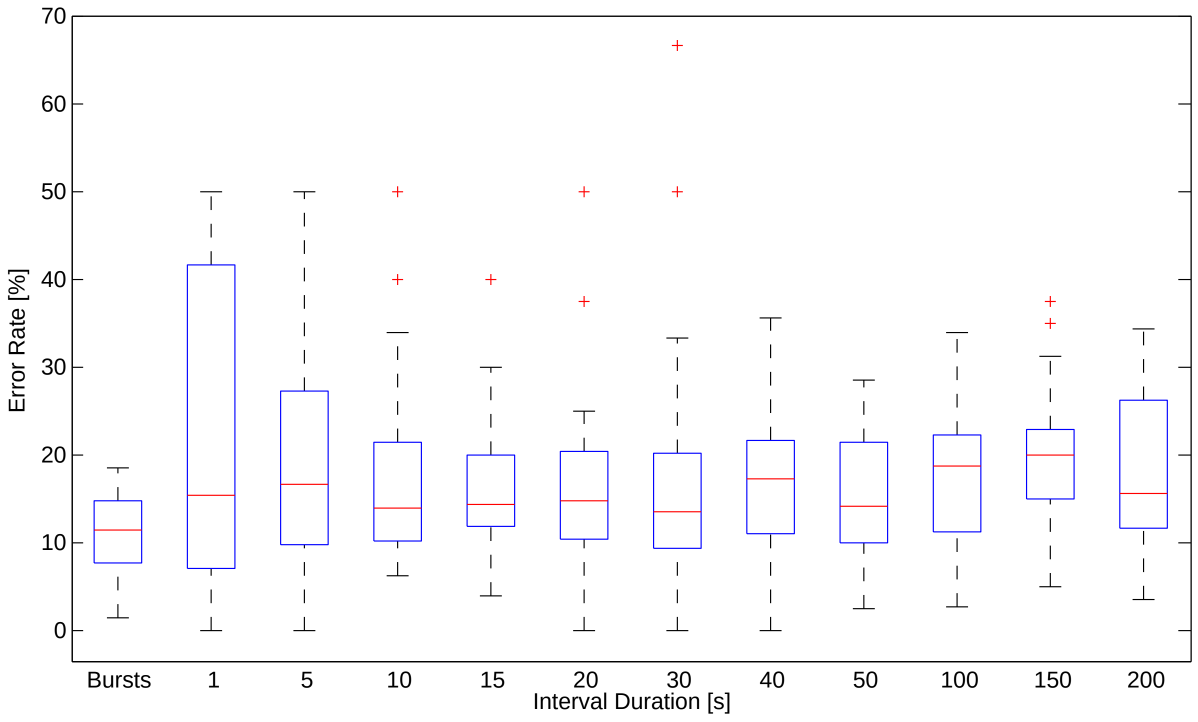

Figure 6.8: Comparing traffic separation into bursts to traffic division into constant time intervals (User dataset).

6.6 Variation of Fingerprinting Duration

In previous experiments, fingerprints were always computed using 70% of the full eight hours of captured traffic from the corresponding user. We consider this setting, namely an adversary being in possession of not arbitrary but sufficient user traffic, to be realistic. An attacker could for instance record transmissions of the target device during night while the user is at home or at any other known location. However, we also want to assess impacts on the classification accuracy if fingerprints are based on

shorter capturing periods. Hence, in this section, we will examine effects of varying fingerprinting durations.
In a first experiment, we built a fingerprint for every user based on the set of bursts out of a t minutes sub-interval from the overall eight hours traffic ($t \in \{10, 30, 60, 90, 120, 180, 240, 360\}$). For each user, the sub-intervals were chosen randomly ten different times. All remaining bursts from the eight hours traffic, effectively the inverse interval, were part of the test set and have been classified individually. For the training and test set, evenly many random bursts from other users were picked in order to turn our problem into a binary classification. Figure 6.9 depicts the resulting classification error rates depending on how many minutes of traffic have been utilised to build the fingerprint. False positive and false negative rates are represented by the green and blue boxes, respectively. As expected, the longer the fingerprinting duration, the lower the error rates. However, especially the false negative rate improves for larger intervals. The false positive rate shows no significant changes from a duration of two hours onwards. In the upper left corner, two outliers appear with error rates of 100%. This can be explained by a selection of a 10 minute interval, where no bursts at all occurred and consequently no fingerprint could be created.

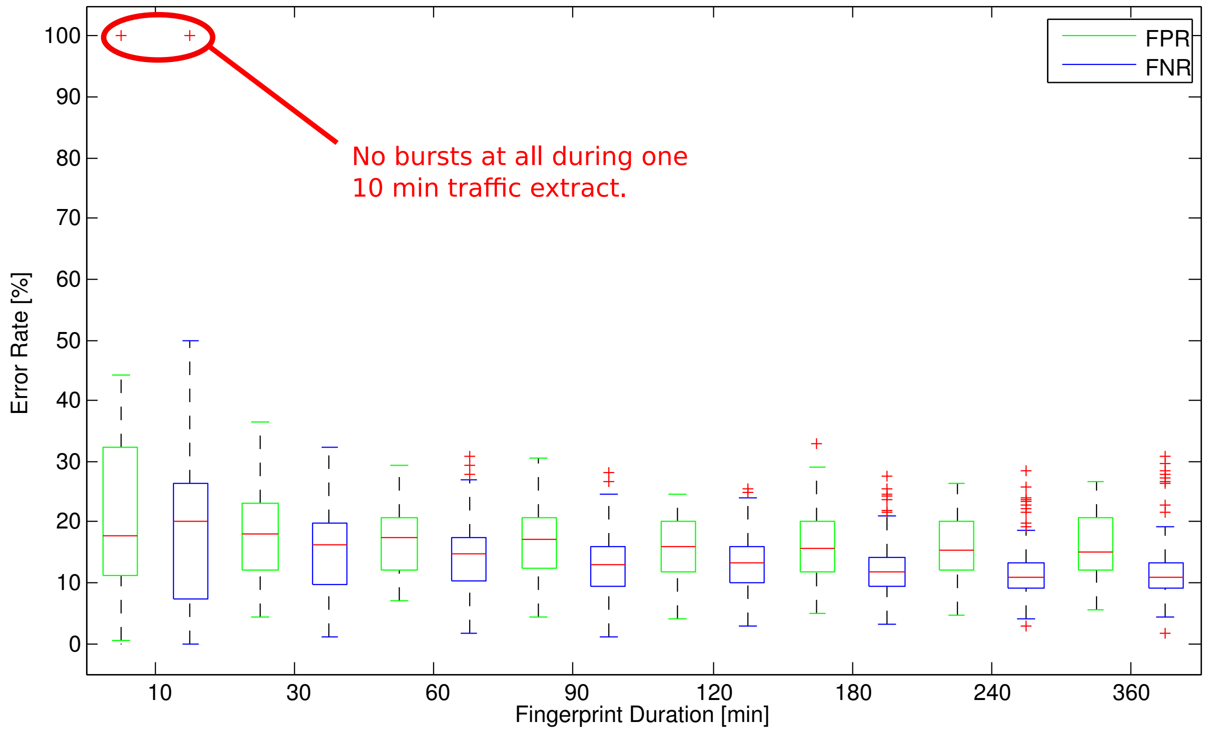

Figure 6.9: Impacts of the fingerprinting duration on the classification accuracy. The longer traffic is captured to build the fingerprint, the lower are the resulting classification error rates.

Figure 6.10 depicts the results for the experiment from Section 6.3 that was conducted for different fingerprint durations. Instead of the inter-quantile ranges, only median error rates (sum of false positive and false negative rate) are plotted. As it can be observed, only the curve representing the full eight hour fingerprint reaches a median error rate of 0% before the number of 40 bursts that have been applied for classification. Due to the very small amount of available bursts in a 10 minute sub-interval, no improvement at all in terms of the error rate can be observed for the upper curve. As seen in Figure 6.9, there even exist 10 minute time slots where not a single burst appears. All in all, this experiment leads to two main conclusions. Firstly, an adversary needs a certain amount of traffic to construct a fingerprint and accomplish a reliable subsequent identification. In our case, for the non-interactive dataset, this implies at least 30 minutes of transmissions. Secondly, the more traffic available for fingerprinting, the less traffic is needed in order to get an accurate classification result afterwards.

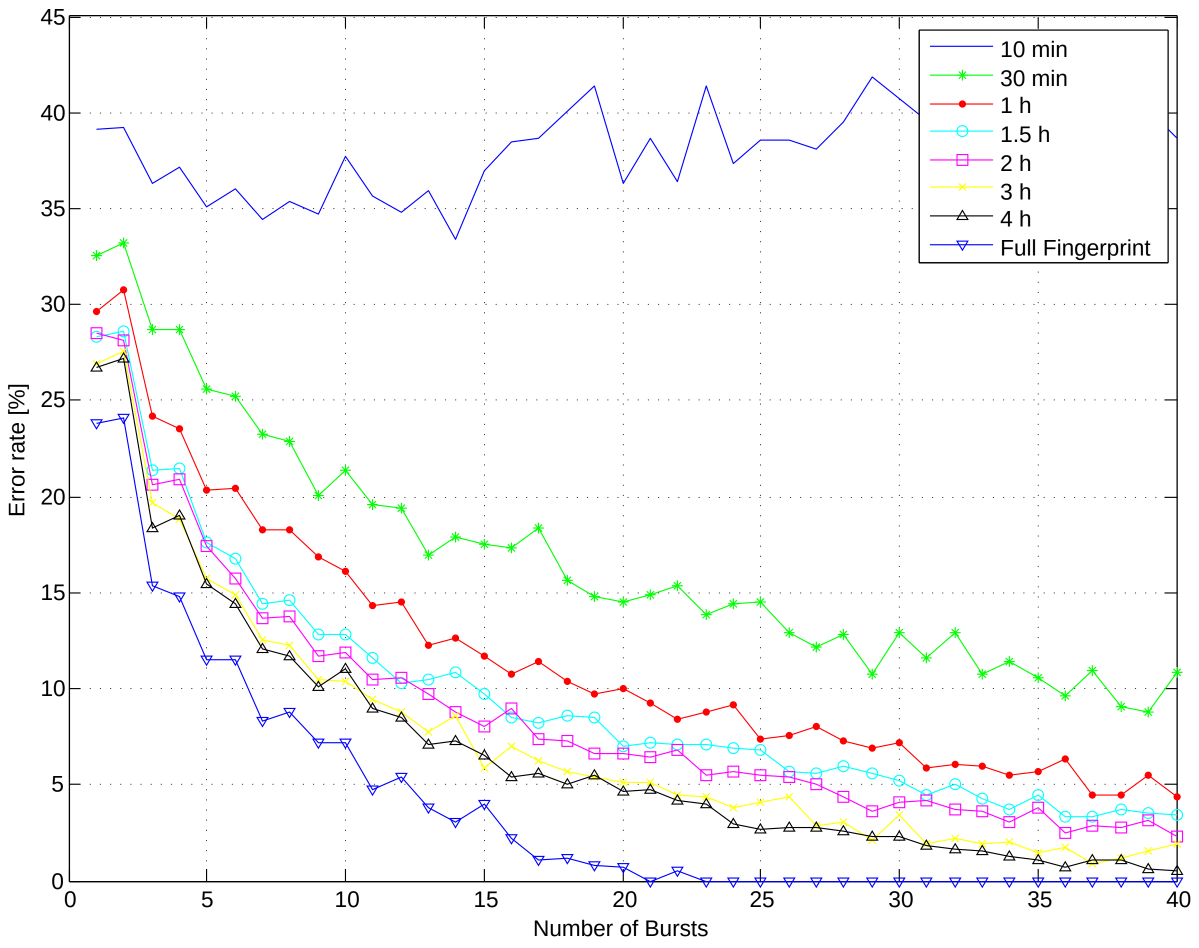

Figure 6.10: Error rate depending on the number of bursts employed for classification. The different curves represent the median error rates for cases with different fingerprint durations.

7 Conclusion and Future Work

7.1 Conclusion

In this work, we investigated the question of whether background traffic generated by smartphone applications can be used as a fingerprint to identify and discriminate different smartphones. In particular, we based the fingerprint features solely on the timing and the amount of transmitted data, i.e., only the features available as side-channel information. They can be easily extracted through monitoring of wireless channels used by the UMTS radio technology and without assuming any knowledge of the payload. Our results show that the multitude of installed applications and their frequent background communication generate a unique behaviour that allows an eavesdropper to accurately identify smartphones. Along this way, we designed extensive experiments including traffic monitoring from the most popular Apps and demonstrated that even if the smartphones have a high number of the same applications installed, they still can be successfully identified with a very high accuracy. In particular, after the fingerprint is generated, the eavesdropper requires only $\sim$15 minutes of the captured traffic to achieve more than 90% classification accuracy. Furthermore, we showed that a reliable identification is even possible by only considering the easier to get hold of downlink traffic. Even traffic-causing user interactions do not significantly impair our classification accuracy.
These results have direct impact on the user's privacy as they justify that an adversary is able to detect whether a smartphone is associated with a certain UMTS radio cell. Based on this knowledge, the attacker is able to estimate the user's current geographical position.

7.2 Future Work

One problem during the data gathering phase of this work was to find probands willing to get their network traffic captured. Even though we do not regard the payloads of transmissions, users tend to have privacy concerns. Hence, in terms of future work, a first important step would be to conduct our experiments on a larger sample set of smartphones that are actually used by real persons, similar to the user dataset (cf. Section 3.1), but on a bigger scale.

The present consideration of the fingerprint quality is of local nature. As stated in Section 4.5.1, even a nearly optimal local quality Q_f can be futile if most of the other smartphones have equally high local qualities with almost the same phone state. Therefore, it would be interesting to get an insight of which App combinations are installed on a vast set of smartphones instead of just having the absolute number of downloads. In this way, it is possible to correlate with fingerprints of other devices and obtain a global view on the fingerprint quality.

Moreover, instead of leaking the user identity through side-channel information, one could try to identify single installed applications on a smartphone. This also results in a privacy violation and could be used as a basis for launching more sophisticated attacks afterwards. For instance an adversary tries to determine whether a certain phone is using a particular App that is currently vulnerable to a remote exploitation attack.

Last but not least, the actual implementation of an UMTS sniffer being capable of demodulating and demultiplexing the 3G signal on the air interface represents another potential task of future work.

Bibliography

[1] Gartner. `http://www.gartner.com/it/page.jsp?id=2120015`, 2012. [Online; accessed 11/10/2012].

[2] Nielsen. `http://tinyurl.com/8y2e773`, 2012. [Online; accessed 11/10/2012].

[3] UMTS link. `http://www.umtslink.at/3g-forum/vbcms.php?area=vbcmsarea_content&contentid=40`, 2012. [Online; accessed 09/10/2012].

[4] 3GPP. TS 25.213, Spreading and modulation (FDD). Technical report, 1999.

[5] A.A.E. Ahmed and I. Traore. Anomaly intrusion detection based on biometrics. In *Information Assurance Workshop, 2005. IAW '05. Proceedings from the Sixth Annual IEEE SMC*, pages 452 – 453, june 2005.

[6] Kasper Bonne Rasmussen and Srdjan Capkun. Implications of radio fingerprinting on the security of sensor networks. In *Security and Privacy in Communications Networks and the Workshops, 2007. SecureComm 2007. Third International Conference on*, pages 331 –340, sept. 2007.

[7] Vladimir Brik, Suman Banerjee, Marco Gruteser, and Sangho Oh. Wireless device identification with radiometric signatures. In *Proceedings of the 14th ACM international conference on Mobile computing and networking*, MobiCom '08, pages 116–127, 2008.

[8] Capkun S. Danev B., Zanetti D. On physical-layer identification of wireless devices. `http://www.syssec.ethz.ch/research/OnPhysId.pdf`, July 2012. [Online; accessed 22/10/2012].

[9] Loh Chin Choong Desmond, Cho Chia Yuan, Tan Chung Pheng, and Ri Seng Lee. Identifying unique devices through wireless fingerprinting. In *Proceedings of the first ACM conference on Wireless network security*, WiSec '08, pages 46–55, 2008.

[10] Jeffrey Erman, Martin Arlitt, and Anirban Mahanti. Traffic classification using clustering algorithms. In *Proceedings of the 2006 SIGCOMM workshop on Mining network data*, MineNet '06, pages 281–286, 2006.

[11] Hossein Falaki, Dimitrios Lymberopoulos, Ratul Mahajan, Srikanth Kandula, and Deborah Estrin. A first look at traffic on smartphones. In *Proceedings of the 10th annual conference on Internet measurement*, IMC '10, pages 281–287, 2010.

[12] Mario Frank, Ralf Biedert, Eugene Ma, Ivan Martinovic, and Dawn Song. Touchalytics: On the applicability of touchscreen input as a behavioral biometric for continuous authentication. July 2012.

[13] Jason Franklin, Damon McCoy, Parisa Tabriz, Vicentiu Neagoe, Jamie Van Randwyk, and Douglas Sicker. Passive data link layer 802.11 wireless device driver fingerprinting. In *Proceedings of the 15th conference on USENIX Security Symposium - Volume 15*, USENIX-SS'06, 2006.

[14] Ryan M. Gerdes, Thomas E. Daniels, Mani Mina, and Steve F. Russell. Device identification via analog signal fingerprinting: A matched filter approach. In *In 144 Proceedings of the Network and Distributed System Security Symposium (NDSS*, page 78, 2006.

[15] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *J. Mach. Learn. Res.*, 3:1157–1182, March 2003.

[16] Frank R. Hampel. The breakdown points of the mean combined with some rejection rules. *Technometrics*, 27(2):95–107, 1985.

[17] Chris Johnson. *Radio Access Networks for UMTS: Principles and Practice*. Wiley Publishing, 2008.

[18] T. Kohno, A. Broido, and K.C. Claffy. Remote physical device fingerprinting. *Dependable and Secure Computing, IEEE Transactions on*, 2(2):93 – 108, april-june 2005.

[19] 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*, CCS '06, pages 255–263, 2006.

[20] T. Scott Saponas, Jonathan Lester, Carl Hartung, Sameer Agarwal, and Tadayoshi Kohno. Devices that tell on you: privacy trends in consumer ubiquitous computing. In *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, SS'07, pages 5:1–5:16, 2007.

[21] Qixiang Sun, D.R. Simon, Yi-Min Wang, W. Russell, V.N. Padmanabhan, and Lili Qiu. Statistical identification of encrypted web browsing traffic. In *Security and Privacy, 2002. Proceedings. 2002 IEEE Symposium on*, pages 19 – 30, 2002.

[22] O. Ureten and N. Serinken. Wireless security through rf fingerprinting. *Electrical and Computer Engineering, Canadian Journal of*, 32(1):27 –33, winter 2007.

[23] Wikipedia. Support vector machine — wikipedia, the free encyclopedia, 2012. [Online; accessed 5-December-2012].

[24] Charles V. Wright, Lucas Ballard, Fabian Monrose, and Gerald M. Masson. Language identification of encrypted voip traffic: Alejandra y roberto or alice and bob? In *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, SS'07, pages 4:1–4:12, 2007.

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

[26] S. Zander, T. Nguyen, and G. Armitage. Automated traffic classification and application identification using machine learning. In *Local Computer Networks, 2005. 30th Anniversary. The IEEE Conference on*, nov. 2005.

[27] 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, pages 59–70, New York, NY, USA, 2011. ACM.

[28] Nan Zheng, Aaron Paloski, and Haining Wang. An efficient user verification system via mouse movements. In *Proceedings of the 18th ACM conference on Computer and communications security*, CCS '11, pages 139–150, New York, NY, USA, 2011. ACM.