2015

Analysis of Smartphone Traffic

Nicholas Luke Ruffing
Cleveland State University

Analysis of Smartphone Traffic

NICHOLAS RUFFING

Bachelor of Science in Computer Engineering

Cleveland State University

May, 2014

submitted in partial fulfillment of the requirements for the degree

Master of Science in Electrical Engineering

at the

CLEVELAND STATE UNIVERSITY

May 2015

We hereby approve this thesis for

Nicholas Ruffing

Candidate for the Master of Science in Electrical Engineering degree

for the

Department of Electrical and Computer Engineering

and the CLEVELAND STATE UNIVERSITY

College of Graduate Studies

Thesis Chairperson, Dr. Ye Zhu

Electrical and Computer Engineering Department & Date

Thesis Committee Member, Dr. Chansu C. Yu

Electrical and Computer Engineering Department & Date

Thesis Committee Member, Dr. Yuanjian Fu

Electrical and Computer Engineering Department & Date

Student's Date of Defense: April 16, 2015

ACKNOWLEDGMENTS

I would first like to thank Dr. Ye Zhu for giving me the opportunity to work on this project. I would also like to thank Dr. Chansu Yu and Dr. Yuanjian Fu for being a part of my committee as well as the remaining faculty at Cleveland State University who helped me to succeed.

Most importantly, I would like to thank my family and friends for all of their support.

Analysis of Smartphone Traffic

NICHOLAS RUFFING

ABSTRACT

Smartphone reconnaissance, the first step to launch security attacks on a target smartphone, enables an adversary to tailor attacks by exploiting the known vulnerabilities of the target system. We investigate smartphone OS identification with encrypted traffic in this paper. Four identification algorithms based on the spectral analysis of the encrypted traffic are proposed. The identification algorithms are designed for high identification accuracy by removing noise frequency components and for high efficiency in terms of computation complexity. We evaluate the identification algorithms with smartphone traffic collected over three months. The experimental results show that the algorithms can identify the smartphone OS accurately. The identification accuracy can reach 100% with only 30 seconds of smartphone traffic.

TABLE OF CONTENTS

Page

ABSTRACT iv

LIST OF TABLES vii

LIST OF FIGURES viii

CHAPTER

I. INTRODUCTION 1

1.1 Motivation 1

1.2 Contributions 3

1.3 Organization 4

II. RELATED WORK 5

2.1 OS Fingerprinting 5

2.2 Reconnaissance through Packet-Content Agnostic Traffic Analysis 7

2.2.1 Website Fingerprinting 7

2.2.2 Inferring Users' Online Activities Through Traffic Analysis 8

2.2.3 Hidden Services 8

2.3 Analysis of Smartphone Traffic 9

III. NETWORK AND THREAT MODEL 11

3.1 Network Model 11

3.2 Threat Model 12

IV. IDENTIFYING SMARTPHONE OPERATING SYSTEMS 14

4.1 Rationale 14

4.2 Identification Framework 16

4.2.1 Spectrum Generation 17

4.2.2 Feature Extraction 18
4.2.3 OS Identification 19
4.3 Full Spectrum 20
4.4 Spectrum Selection 22
4.5 Suppression 25
4.6 Hybrid Algorithm 29
4.7 Comparison of the OS Identification Algorithms 30
V. EMPIRICAL EVALUATION 33
5.1 Experiment Setup 33
5.2 Performance Metrics 35
5.3 Length of Traffic Traces 35
5.4 Number of Labeled Traces 39
5.5 Sample Interval 40
5.6 Number of Top Significant Frequency Components Kept (Suppression Algorithm) 41
VI. DISCUSSION 44
6.1 Application Identification 44
6.2 Running Time 45
VII. CONCLUSION 47
BIBLIOGRAPHY 49

LIST OF TABLES

Table Page

I Complexity of the Feature Extraction and Correlation in the Identification Algorithms (P: Number of Different Smartphone OSes, Q: Number of labeled traces available for each smartphone OS, K: Number of most significant frequency components to keep, L: Length of Spectra) 31

II Specifications of Smartphones Used 34

III Application Resource Consumption (The consumption data is obtained by running each application alone for 50 minutes on the devices. Since the Symbian OS and the Windows phone do not have built-in monitoring tools, we do not collect the data for the two OSes.) 36

IV Empirical running times for the Full Spectrum, Suppression, Spectrum Selection and Hybrid OS detection algorithms. 15 minutes of traffic with an 8 ms sample interval is used to generate this data. 45

LIST OF FIGURES

Figure		Page
1	A sample frequency spectrum of YouTube streaming traffic on the Android OS. To generate the spectrum 50 minutes of streaming traffic with an 8 ms sample interval is used.	16
2	Correspondence Between the periodicities in a Time Domain Signal and the Characteristic Frequency Component in the spectrum	17
3	Identification Framework Diagram	17
4	Magnitude Distribution of the Frequency Spectrum in Figure 1	26
5	Data Collection Setup	34
6	Identification Performance with Traces of Different Length	36
7	False Alarm Rates (Spectrum Selection Algorithm)	39
8	Identification Performance with Different Number of Labeled Traces .	39
9	Identification Performance with Different Sample Intervals	40
10	Number of Top Significant Frequency Components to Keep in the Suppression Algorithm. K is determined in the feature selection step (Algorithm 7) based on the mean and standard deviation of the magnitude of frequency components in the spectra of the labeled traces.	42
11	Application Identification Performance (Spectrum Selection). The applications to be detected are YouTube, Skype, and Downloading. The applications are equally distributed making the random guess rate 33%.	45

CHAPTER I

INTRODUCTION

1.1 Motivation

This paper studies the identification of operating systems (OS) of smartphones that communicate using encrypted traffic. Smartphones have become the central communication and computing devices in our daily life because of (a) their nearly ubiquitous Internet access through various communication capabilities such as WiFi, 3G, or even 4G networks, (b) their user-friendly interfaces supporting touch and gesture based input, and (c) their numerous applications and games. With the increasing reliance on smartphones, users are increasingly using them to share sensitive data, such as personal contacts and banking information. Smartphones are also adopted in business and military environments [9] because of their portability and constant network access. As a result, smartphone security is of great importance nowadays.

In order to launch an effective attack on a particular smartphone an attacker must be able to tailor the attack to the target smartphone's platform (i.e. OS). This is turn requires that the attacker be able to identify the operating system running on the

target smartphone. Once the attacker knows the target OS, he or she becomes able to exploit known vulnerabilities both of the smartphone OS and of the applications and services running on the OS. The most readily obtainable information that enables OS identification is the wireless traffic generated by the target smartphone. Since more and more smartphone traffic is encrypted to protect the confidentiality of the wireless communications [7], the OS identification must not rely on the content of packets or packet headers.

The ability to identify smartphone OSes has many applications, some of which are malicious in nature while others are not: (1) As a smartphone owner or a smartphone defense designer, we would like to know how susceptible a particular OS platform is to identification based on encrypted traffic. (2) The OS identification can enable content providers, including websites, to tailor the content for different applications running on smartphones in different OSes. (3) The OS identification in conjunction with application identification enables network operators, especially mobile network operators, to predict the bandwidth requirements from a smartphone so that the network operators can better allocate resources to match expected bandwidth requirements.

When the traffic is encrypted, the observer can not access packet content, and his or her ability to monitor the traffic is limited to the timing of the packets. Observations indicate, however, that different OSes cause the smartphone to generate traffic with different timing. Differences in timing footprints are caused by differences in OS implementations (e.g. CPU scheduling, TCP/IP protocol stack), and by differences in resource management (e.g. memory management or power management). Similarly, differences in applications caused by the OS differences (e.g. audio/video codecs available for multimedia communications) become visible in the timing footprint of sent packets as well.

1.2 Contributions

In this paper we describe how differences in OSes can be identified by analyzing the timing traces of the generated traffic in the frequency domain. Frequency domain analysis is a classical tool to analyze temporal signals [18], including the timing behavior of traffic in our project, by converting signals from the time domain to the frequency domain.

The main challenge in OS identification with the frequency analysis comes from the fact that the frequency spectrum contains many *noise frequency components*, i.e., frequency components that are not caused by the OS features, but rather by application or user behavior. The noise frequency components can be caused by network dynamics (such as network congestion and round trip time), and traffic content (such as periodicities in the video content when streaming a video clip). In this paper, we call the frequency components that are helpful for OS identification, i.e., the frequency components caused by OS features, the *characteristic frequency components*. The effectiveness of any frequency-domain based identification clearly depends on its ability to filter out noise and keep the *characteristic frequency components*.

Once the frequency spectrum of a device has been collected, it must be matched against training data, that is the spectrum of interest needs to be matched with the spectrum generated by a known smartphone OS. Correlation can be used for this matching. The complexity of the matching is $O(L)$ where L denotes the length of the spectra. In this paper, we propose approaches to reduce the computational complexity significantly and still identify smartphone OS accurately.

Our major contributions are summarized as follows:

- We propose four smartphone OS identification algorithms. The identification algorithms use frequency spectrum to capture the differences in smartphone OSes. Correlation is used to match the spectrum of interest to the spectra

generated by known smartphone OSes.

- We propose OS identifications algorithms that can remove noise frequency components to improve the identification accuracy.

- We evaluated the OS identification algorithms with extensive empirical experiments. The experiments with 336 GB of smartphone traffic collected over 3 months show that the OS identification algorithm can identify smartphone OS with very high accuracy and only small amounts of smartphone traffic.

- We extend the OS identification algorithms to identify the applications running on smartphones. We applied the application identification algorithms to identify popular applications available on smartphones in different OSes. The experimental results show that high identification accuracy can be achieved with as little as 30 seconds of smartphone traffic.

1.3 Organization

This paper is organized as follows: Chapter II reviews related work. The network model and the threat model used in this paper are presented in Chapter III. We explain the rationale behind the proposed identification approach and describe the details of the smartphone OS identification algorithms in Chapter IV. In Chapter V, we evaluate the smartphone OS identification algorithms with 336 GB of traffic data collected over 3 months. The extension of the OS identification algorithms for application identification is discussed in Chapter VI. We conclude the paper in Chapter VII with a discussion of the future work.

CHAPTER II

RELATED WORK

In this chapter, we review related work on existing OS fingerprinting approaches, reconnaissance through traffic analysis, and analysis of smartphone traffic.

2.1 OS Fingerprinting

Approaches to traffic based fingerprinting can be either passive or active. In the former the observer monitors the traffic from the target, while in the latter the observer may stimulate the target by sending requests and cause the target to display a richer behavior for the observer to monitor. Most existing passive methods for computer OS fingerprinting are based on packet headers. The methods discussed in [16] detect the computer OS by checking the initial Time to Live (TTL) value in the IP header and the TCP window size in the first TCP packet. Methods to identify the computer OS by inspecting the application layer data in traffic, such as server banners in HTTP, SSH, and FTP as well as HTTP client User-Agent strings are also discussed in [16]. Kollmann [13] proposes to fingerprint the computer OS based

on its implementation of the DHCP protocol, as different computer OSes support different combinations of DHCP options. The network analysis tools `siphone` and `p0f` developed as a part of the Honeynet Project [21] fingerprint the computer OS by checking four TCP signatures. Two of them are the TTL and the TCP window size as discussed in [16]. The two additional signatures are on the Don't Fragment (DF) bit and the Type of Service (ToS) bits.

Active OS fingerprinting methods to identify the OS of a remote machine are used by Nmap [17], a software utility for network discovery and security auditing. Nmap identifies the remote OS by sending TCP/IP probes and checking how the remote machine responds to these probe packets. Based on the response, Nmap uses its large database of heuristics to identify the OS. Another software package created by Durumeric *et al.* [5], called Zmap, allows a single computer with a gigabit Ethernet connection to scan the entire IPv4 address space in 45 minutes. This is a new tool that can easily be used to establish active fingerprints very quickly.

Countermeasures have been proposed that are designed to defeat OS fingerprinting. Smart *et al.* [22] developed a TCP/IP stack fingerprint scrubber to defend against active and passive OS fingerprinting attacks based on the TCP/IP stack. The scrubber sanitizes packets from a group of hosts at both the network and transport layers to block fingerprinting scans. These sanitized packets will not reveal OS information.

All of the computer OS fingerprinting methods reviewed above require access to the packet headers or packet content. As a result these methods are largely ineffective when applied to encrypted traffic.

2.2 Reconnaissance through Packet-Content Agnostic Traffic Analysis

Various reconnaissance approaches through packet-content agnostic traffic analysis have been proposed, and some of the approaches are studied in the context of privacy breaches.

2.2.1 Website Fingerprinting

Herrmann *et al.* [11] developed a method for website fingerprinting with traffic encrypted and anonymized by Tor. The method uses common text mining approaches on frequency distributions of packet sizes. The method is reported to be capable of identifying 300,000 real-world traffic traces with 97% accuracy using a sample of 775 sites. Panchenko *et al.* [20] showed the effectiveness of website fingerprinting attacks on anonymity networks. Their approaches can increase the detection accuracy from 3% to 55% with a Tor data set and from 20% to 80% with a JAP data set. Their experiments on a real-world data set can achieve an accuracy of 73%. The countermeasure of applying camouflage to hamper the fingerprinting attack was proposed in [20] and the countermeasure is able to decrease the accuracy to as low as 3%. A website detection attack that could be executed from a remote location was proposed in [8]. The attack first estimates the load inside a victim's router queue by sending regularly spaced probe packets and then measuring their round trip time. Based on this estimation of the load, any of the website fingerprinting methods described above can be used. Cai *et al.* [2] attempted to defeat countermeasures proposed for website fingerprinting, more specifically HTTPOS and randomized pipelining over Tor. The method used packet size vectors from encrypted traffic and the Damerau-Levenshtein algorithm to detect which web pages the traffic is associated with. They were able to

achieve website fingerprinting accuracy as high as 90% against some countermeasures with a sample set of 100 websites. Early on, Liberatore and Levine [14] proposed traffic analysis on encrypted HTTP streams to infer the source of a web page retrieved in encrypted HTTP streams. A profile of each known website is created in advance. The traffic analysis identifies the source by comparing observed traffic with established profiles with classified algorithms. They used a sample size of 2,000 websites with 400,000 traffic traces.

2.2.2 Inferring Users' Online Activities Through Traffic Analysis

Zhang *et al.* [25] use short traces of encrypted traffic on IEEE 802.11 wireless local area networks (WLAN) to infer activities of a specific user (e.g. web browsing, file downloading, or video streaming). Their experiments include traffic traces from web browsing, online chatting, online gaming, file downloading, and video conversations. They developed a learning hierarchical classification system to discover if web activities were associated with a traffic trace. They performed their experiments in a home environment, a university campus, and on a public network. They were able to infer the users activities with 80% accuracy using 5 seconds of traffic and 90% accuracy with 1 minute of traffic.

2.2.3 Hidden Services

Hidden services are used in anonymity networks like Tor to resist censorship and attacks like a denial of service attack. Øverlier and Syverson [19] propose attacks that reveal the location of a hidden server in the Tor network. Using one corrupt Tor node they were able to locate a hidden server in minutes. They then proposed changes to the Tor network in order to resist their attacks and these changes were

implemented. A very similar effort in [1] investigates the flaws in the Tor network and its hidden services. Three practical cases including a botnet with hidden services for command and control channels, a hidden service used to sell drugs, and the DuckDuckGo search engine are used for evaluation. Their method involves first gaining control of the descriptors of a hidden service and then performing a traffic correlation attack on the hidden service. Zander and Murdoch [24] aim to improve their clock-skew measurement technique for revealing hidden services. Their original method [15] correlates clock-skew changes during times of high load. They noticed two areas of noise, network jitter and timestamp quantization error, and aim to reduce the latter by synchronizing measurements to the clock ticks. They were able to reduce the timestamp quantization error and increase their accuracy by two magnitudes.

2.3 Analysis of Smartphone Traffic

Smartphone traffic has been analyzed for various purposes. In [23] Tzagkarakis *et al.* proposed to use the Singular Spectrum Analysis to characterize network load in a large WLAN. This is beneficial to monitor the load and to place access points accordingly. Their findings can help design large-scale WLAN's that can be used by smartphones in large public areas. Chen *et al.* [4] studied the network performance of smartphones in a university-wide WLAN. They analyzed 2.9 TB of data collected over three days and were able to gather interesting insights on TCP and application behavior of smartphones and their effect on performance. Huang *et al.* [12] proposed a methodology for comparing application performance based on 3G communications. Their study shows how YouTube buffering techniques vary across smartphone OSes.

Throughout these experiments, the observer has access to header and payload data. In comparison, our work is focusing on the analysis of encrypted traffic. To our knowledge, we are describing in this paper the first work proposing traffic analysis of

encrypted traffic for smartphone OS identification.

CHAPTER III

NETWORK AND THREAT MODEL

3.1 Network Model

In this paper, our goal is to identify smartphone operation systems (OS) when the smartphone communicates using encrypted traffic. The capability of OS identification is needed for various purposes: (1) On one hand, to launch attacks to a smartphone, an attacker needs to determine first the OS and then the applications running on the target smartphone. Given the OS and application information, attacks can exploit known vulnerabilities to tailor attacks specific to the OS and the applications. On the other hand, to defend against the reconnaissance from the attackers, smartphone defense designers and smartphone owners need to know how accurate the identification can be. (2) The OS identification can enable content providers, including websites, to tailor the content for different applications running on smartphones in different OSes. (3) The OS identification allows mobile network operators to predict the bandwidth requirements from any particular smartphone so that the network operators can better allocate resources with the knowledge of expected bandwidth

requirements.

We are particularly interested in the identification based on WiFi traffic for three reasons: First, although current smartphones have various communication capabilities, such as WiFi, 3G, or even 4G, nearly every smartphone on the market is capable of WiFi communication. Next, the majority of traffic from smartphones is sent through WiFi [3] partly because of its low cost and relatively high bandwidth. Finally, WiFi based passive attacks are easy to stage. A drive-by or walk-by detection of the smartphone OS is therefore very easy to stage.

3.2 Threat Model

In this paper we assume a passive adversary who is able to capture packets exchanged by the smartphone of interest. The smartphone, in turn, communicates using encryption. This assumption reflects the increasing popularity of encryption tools available for smartphones [10]. The traffic encryption disables access to packet content and renders traffic analysis based on packet content ineffective.

In summary, we assume that the adversary has the following capabilities:

- The adversary is able to eavesdrop on WiFi communications from the target smartphones and collect encrypted traffic for the identification.
- The adversary is able to collect traffic from known smartphone OSes and analyze the traffic for future identification.
- We assume a passive adversary. That is, the adversary is not allowed to add, delete, delay, or modify existing traffic for OS identification.
- The traffic traces including the traffic traces collected for training on known smartphone OSes and the traffic traces of interest for identification by the adver-

sary are collected independently. In other words, the traffic traces are collected in different network sessions and possibly on different WiFi networks.

Other attack scenarios can be very easily imagined. For example one where the observer does not have access to the wireless link, but rather collects data on the wired part of the path downstream. In this paper we focus on data collection on the wireless link.

CHAPTER IV

IDENTIFYING SMARTPHONE OPERATING SYSTEMS

In this chapter, we present approaches to identify smartphone operating systems based on the threat model described above. We begin this chapter with an introduction of the rationale behind the identification approach. We then describe the identification framework and the details of each identification algorithm.

4.1 Rationale

OS identification through encrypted traffic is possible because of implementation differences and differing resource management policies among smartphone OSes. These differences include:

- Differences in OS implementations: Different smartphone OSes may have different kernels, different CPU scheduling algorithms, and different implementations of the TCP/IP protocol stack. These differences in the OS implementations can cause the timing behavior of traffic to differ from one smartphone OS to another.

- Differences in Resource Control: Smartphones are resource-constrained devices. Largely due to their small form factor, smartphones have limited CPU processing capability, limited memory, and limited battery lifetime. To better utilize these resources, smartphone OSes adopt a number of policies for resource control. For example, different smartphone OSes may have different power management policies.

- Differences in Applications: Because of the differences across OSes, the same application for different smartphone OSes may be implemented differently. For example, different OSes support different combinations of audio and video codecs used for multimedia communications. Obviously, different codecs used for multimedia communications will very likely generate network traffic differently. Another example is YouTube: In [12], iPhone is reported to first download a portion of video at a high rate, pause a while, and then continue downloading. The authors conjecture that this pattern is caused by the memory management and power saving policy in Apple's iOS. The Android phone reported in [12] periodically downloads small chunks of YouTube video every 10 seconds. The authors conjecture is that the download pattern is due to hedging against the user not wanting to watch the entire video.

The differences described above will obviously give raise to different timing behaviors for the traffic generated by different smartphone OSes. These differences in the timing behavior can be easily captured in the frequency domain. A typical spectrum of YouTube video streaming on the Android OS is shown in Figure 1: We can observe that the YouTube traffic flow has many significant frequency components. While some of these components are coincidental, others are associated with the YouTube buffering strategy on the Android OS. Others again may be associated with specific OS implementation approaches. To show the correspondence, we draw

the time domain signal of the YouTube traffic flow in Figure 2(a). We can easily observe the periodic nature of the buffering in the figure. By checking the data, we confirm that the periodic buffering happens every 250 seconds. For verification, we zoom in the corresponding frequency range of Figure 1 and the zoomed-in portion is shown in Figure 2(b). In Figure 2(b), we can observe the peak at the frequency of 0.004Hz, which corresponds to the buffering period of 250 seconds. Obviously the frequency component corresponding to the buffering is helpful in OS identification. We call such frequency components *characteristic frequency components*. In Figure 1, we observe a large number of *noise frequency components* as well, which in turn are caused by network dynamics such as round trip time and the video content. These noise frequency components are not caused by OS features, and they are therefore not helpful for OS identification. Obviously, removing the *noise frequency components* will very likely improve the identification accuracy.

4.2 Identification Framework

We propose four identification algorithms for OS identification. The four identification algorithms are designed under the same framework. So before introducing

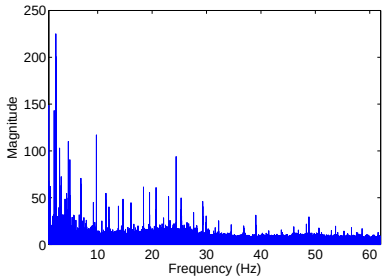
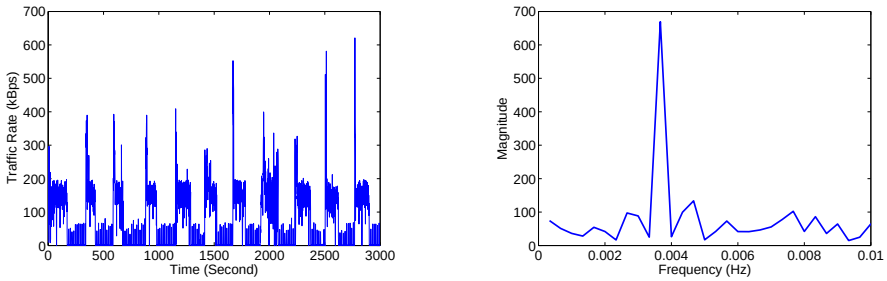


Figure 1: A sample frequency spectrum of YouTube streaming traffic on the Android OS. To generate the spectrum 50 minutes of streaming traffic with an 8 ms sample interval is used.



(a) Time domain signal of the YouTube streaming traffic on the Android OS.

(b) Zoomed-in Portion of the Spectrum shown in Figure 1

Figure 2: Correspondence Between the periodicities in a Time Domain Signal and the Characteristic Frequency Component in the spectrum

the details of each identification algorithms, we present the framework first.

The identification can be divided into two phases as shown in Figure 3: *training phase* and *identification phase*. The training phase consists of two steps: *spectrum generation* and *feature extraction*. The two steps in the identification phase are *spectrum generation* and *OS identification*. We describe the details of each step below.

4.2.1 Spectrum Generation

The spectrum generation step converts traffic traces into frequency spectra. The input of this step is a vector $S = [s_1, s_2, \cdots, s_N]$, where N is the number of samples. The element s_i in the vector is calculated as follows:

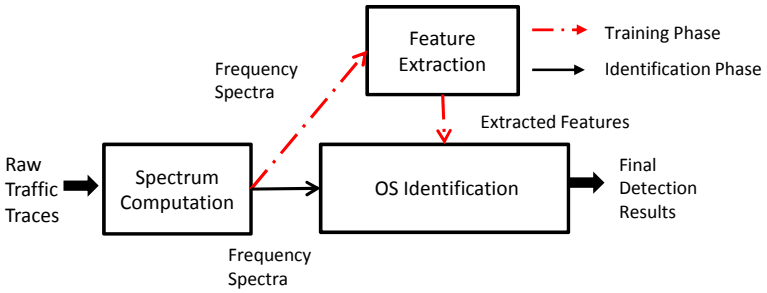


Figure 3: Identification Framework Diagram

$$s_i = \frac{sum\ of\ bytes\ received\ during\ the\ ith\ sample\ interval}{T} \quad , \tag{4.1}$$

where T is the length of sample intervals. The output of this step is the corresponding frequency spectrum $F^S = [f_1^S, f_2^S, \cdots, f_M^S]$, where M denotes the length of the spectrum. The spectrum F^S is calculated in two steps. First we apply the Discrete Fourier Transform (DFT) to the vector S as follows:

$$y_k = \sum_{j=1}^{N} s_j \omega_N^{(j-1)(k-1)}, k = [1, 2, \cdots, M] \quad , \tag{4.2}$$

where y_k denotes the transform coefficients, $\omega_N = e^{-\frac{2\pi i}{N}}$[1], and N denotes the number of samples. The spectrum F^S is calculated as below:

$$f_k^S = |y_k|, k = [1, 2, \cdots, M] \tag{4.3}$$

where the operator $|\cdot|$ denotes the absolute value. Because of the symmetry of the spectrum [18], we only use the single-sided spectrum, i.e., $F^S = [f_1^S, f_2^S, \cdots, f_L^S]$ where $L = \lfloor \frac{M}{2} \rfloor + 1$.

The spectrum generated in this step is fed to the feature extraction step in the training phase or fed to the OS identification step in the identification phase.

4.2.2 Feature Extraction

The feature extraction step is designed to extract features in the frequency spectra generated in the previous step for OS identification. The inputs to the step are the frequency spectra of *labeled traces* that we use for training. The outputs are the features that are selected for the identification step.

[1] We use i to denote the imaginary unit.

4.2.3 OS Identification

The identification step identifies the OS based on two inputs: (1) F^x, the spectrum generated from the trace of interest, denoted as Trace x, and (2) the feature selection from the feature extraction step in the training phase. The output will be the identification result. The pseudo code of the OS identification step for each identification algorithm can be found in the remainder of this chapter.

In all the four algorithms, the OS identification step will first apply the feature selection decided in the feature extraction step to F^x, the spectrum generated from the trace of interest. We denote the feature extracted spectrum as F'^x. The selected spectral features of the test trace will be compared with the spectral features in the labeled traces of each smartphone OS by correlation. In the following, we denote the pth labeled trace of smartphone OS A as A_p, its spectrum as F^{A_p}, and its feature-extracted spectrum as F'^{A_p}. The correlation between the two feature-extracted spectra F'^x and F'^{A_p} can be calculated as follows:

$$corr(F'^x, F^{A_p}) = \frac{\sum_{k=1}^{L} (f_k'^x - \overline{F'^x})(f_k'^{A_p} - \overline{F'^{A_p}})}{\sqrt{\sum_{k=1}^{L} (f_k'^x - \overline{F'^x})^2 \sum_{k=1}^{L} (f_k'^{A_p} - \overline{F'^{A_p}})^2}} \quad , \tag{4.4}$$

where $\overline{F'^x} = \frac{\sum_{k=1}^{L} f_k'^x}{L}$ and $\overline{F'^{A_p}} = \frac{\sum_{k=1}^{L} f_k'^{A_p}}{L}$.

The identification decision is made by comparing the average of the correlation between F'^x, the feature-extracted spectrum of the test trace, and all the feature-extracted spectra of labeled traces generated by the same smartphone OS. We denote the average of the correlation between the trace x and the labeled traces generated by smartphone OS A as $corr_A$. If the average correlation $corr_A$ is the largest among the average correlations between trace x and the labeled traces generated by any smartphone OS, the identification step declares the Trace x to match smartphone OS

A.

In the following we describe and compare four identification algorithms, which we call full spectrum, spectrum selection, suppression, and hybrid. The framework described above is used in designing all four identification algorithms proposed in this paper: The spectrum generation step is the same in all four algorithms. The major differences among the four identification algorithms are in how features are extracted during training and how the extracted features are in turn used for identification.

4.3 Full Spectrum

Function 1: Feature Extraction (Full Spectrum)

Input: $F^{p,q}$: The qth spectrum generated by the pth smartphone OS, $1 \leq p \leq P$, $1 \leq q \leq Q$, where P and Q denote the number of different smartphone OSes and the number of traces available for each smartphone OS respectively

Output: $F'^{p,q}$: The qth feature-extracted spectrum of the pth smartphone OS, $1 \leq p \leq P$, $1 \leq q \leq Q$

```
for p ← 1 to P do
    for q ← 1 to Q do
        for k ← 1 to L − 1 do
            f'^{p,q}_k = f^{p,q}_{k+1};
            // just remove the DC component from the spectrum,
               L: Length of the spectra
        end
    end
end
```

The full spectrum identification algorithm includes all frequency components except for the DC component as feature frequency components. We remove the DC component because the DC component is largely determined by the average traffic rate, which depends on the content in the traffic instead of the smartphone OS.

The spectrum generation step of the full spectrum identification algorithm is the same as the step described in Section 4.2. The pseudo code of the feature extrac-

Function 2: OS Identification (Full Spectrum)

Input: $F'^{p,q}$: Feature extracted spectrum of each labeled trace, F^x: spectrum of the test trace

Output: OStype: Smartphone OS Type

`// remove the DC component from the spectrum` F^x

for $k \leftarrow 1$ **to** $L-1$ **do**

  $f'^x_k = f^x_{k+1}$;

end

OStype = Decision($F'^{p,q}$, F'^x);

Function 3: Decision

Input: $F'^{p,q}$: Feature extracted spectrum of each labeled trace, F'^x: Feature extracted spectrum of the test trace

Output: OStype: Smartphone OS Type

`// correlate the feature-extracted spectrum of the test trace with the feature-extracted spectra of the labeled traces`

for $p \leftarrow 1$ **to** P **do**

  for $q \leftarrow 1$ **to** Q **do**

    Calculate $corr_{p,q}$ the correlation between F'^x and $F'^{p,q}$;

  end

  $corr_p = \frac{\sum_{q=1}^{Q} corr_{p,q}}{Q}$;

  `// average correlation between` F'^x `and the feature-extracted spectra of OS type` p

end

Find the maximum $corr_k$ from the vector $[corr_1, corr_2, \cdots, corr_P]$;

`// without loss of generality, we assume the maximum is` $corr_k$ `in the vector`

OStype = k;

tion step and the OS identification step can be found in Function 1 and Function 2, respectively.

4.4 Spectrum Selection

The spectrum selection identification algorithm is designed to improve the identification performance by removing *noise frequency components*, which are not helpful for OS identification, from the spectrum. As shown in Figure 1 and Figure 2, a traffic flow may have many frequency components. The frequency components include *characteristic frequency components*, such as the frequency components caused by the OS's power management, as well as *noise frequency components*, such as the frequency components caused by network round-trip time, network congestion, and other effects caused by network dynamics. Obviously, removing the noise frequency components can improve identification performance.

Ideally, each frequency component should be evaluated to decide whether it is helpful for OS identification. But the computational cost is prohibitive because of the large number of possible combinations. To make this approach practical, we apply a genetic algorithm to decide which frequency components should be kept for OS identification.

Whether a frequency component is helpful for OS identification is decided during the training phase, based on the labeled traces. The feature extraction step will first divide the labeled traces into two sets: Set_A and Set_B. Instead of exhaustively searching over all the possible combinations of selected frequency components, the step searches for the best combination of the selected frequency components by formulating the search as an optimization problem. The objective function to be optimized is the identification rate obtained by identifying the labeled traces in Set_B. The variables of the optimization problem are binary numbers and each of these bi-

Function 4: Feature Extraction (Spectrum Selection)

Input: $F^{p,q}$: The qth spectrum generated by the pth smartphone OS, $1 \leq p \leq P$, $1 \leq q \leq Q$, where P and Q denote the number of different smartphone OSes, the number of traces available for each smartphone OS, and L: number of frequency components

Output: $F'^{p,q}$: The qth feature-extracted spectrum of the pth smartphone OS, $1 \leq p \leq P$, $1 \leq q \leq Q$, $B_{selected}$=[b_1,b_2,$\cdots$,b_L]: spectrum selection vector where the binary bit b_i indicate whether the ith frequency component is selected

$B_{selected}$ = ga(fitfun, Set_a, Set_b);

// We use ga to represent any genetic algorithm and ga accepts the definition of the fitness function $fitfun$ and outputs values of the variables (in our case the vector $B_{selected}$) resulting the maximum of the fitness function. The fitness function $fitfun$ is defined in Function 5

foreach *spectrum in the input* **do**

  for $i \leftarrow$ 1**to**L **do**

    if B_i==*1* **then**

      include the ith frequency component in $F^{p,q}$ to the feature-extracted spectrum $F'^{p,q}$;

    end

    // without loss of generality, we assume the spectrum $F^{p,q}$ is being processed

  end

end

Function 5: Fitness Function (*fitfun*) (Spectrum Selection)

Input: $B_{selected} = [b_1, b_2, \cdots, b_L]$: spectrum selection vector, Set_A: one set of labeled traces, Set_B: one set of the remaining labeled traces

Output: $Rate_{Identification}$: Identification Rate

foreach *spectrum in* Set_A **do**
  for $i \leftarrow 1$**to**L **do**
    if B_i==*1* **then**
      include the ith frequency component in $F^{p,q}$ to the feature-extracted spectrum $F'^{p,q}$;
    end
    `// without loss of generality, we assume the spectrum` $F^{p,q}$ `is being processed`
  end
end
foreach *spectrum in* Set_A **do**
  include the corresponding feature-extracted spectrum into F'^{Set_a};
end
success=0;
foreach *spectrum* $F^{u,v}$ *in* Set_B **do**
  $OStype = OS\ Identification_{SpectrumSelection}(F'^{Set_A}, F^{u,v}, B_{selected})$;
  if $OStype == u$ **then**
    $success = success + 1$;
  end
end
$Rate_{Identification} = \frac{success}{number\ of\ traces\ in\ Set_B}$;

nary numbers indicates whether the corresponding frequency component is selected. We represent the binary variables as a vector $B_{selected} = [b_1, b_2, \cdots, b_L]$ where the binary variable b_i indicates whether the ith frequency component is selected. We use a genetic algorithm to solve the optimization problem. In comparison with the exhaustive search, this approach is more efficient at the cost of possibly finding a local maximum and so leading to a less effective identification.

The pseudo code of the feature extraction step in the spectrum selection algorithm is shown in Function 4 and the pseudo code of the OS identification step is shown in Function 6.

Function 6: OS Identification (Spectrum Selection)

Input: $F'^{p,q}$: Feature-extracted spectrum of each labeled trace, F^x: spectrum of the test trace, $B_{selected} = [b_1, b_2, \cdots, B_L]$: spectrum selection vector
Output: OStype: Smartphone OS Type
for $i \leftarrow 1$ **to** L **do**
  if $b_i = 1$ **then**
    Include the ith frequency component in F^x to the feature-extracted spectrum F'^x;
  end
end
OStype = Decision($F'^{p,q}$, F'^x);
`// The Decision function is defined in Function 3`

4.5 Suppression

The cost of selecting the characteristic frequency components in the selection algorithm is very high, and so we must find more efficient means for feature extraction. The suppression identification algorithm is designed to remove noise frequency components based on two observations that can be made from the typical spectrum shown in Figure 1: (1) Most frequency components in a spectrum are insignificant (i.e. small in magnitude). (2) Most insignificant frequency components are noise fre-

quency components. Based on the two observations, we design an algorithm (which we call suppression algorithm) that suppresses insignificant frequency components and leaves only significant frequency components for OS identification.

A key consideration in designing such an algorithm is the *number* of most significant frequency components to keep for OS identification. To determine this number, we analyze the distributions of the magnitude of all the frequency components in a spectrum. A typical distribution is shown in Figure 4. We observe that (1) the majority of the frequency components are insignificant and (2) the insignificant components' magnitude can be modeled as a Gaussian distribution.

Based on the observations described above, the number of significant frequency components to keep, denoted by K, is determined by the distribution of the magnitude of frequency components. For any given spectrum F^i we first calculate the mean μ_i and standard deviation σ_i of the distribution on the magnitude. We then set a threshold corresponding to $T_i = \mu_i + 6\sigma_i$ making the probability of the magnitude being larger than the threshold T_i to be about 2E-9 according to the property of Gaussian distributions. In other words, the threshold T_i can filter out over 99% of insignificant frequency components. With the threshold calculated for each trace, we find the number of frequency components with magnitude larger than the threshold in each trace. We set K, the number of significant frequency components to keep, to

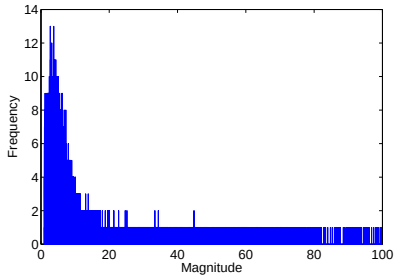


Figure 4: Magnitude Distribution of the Frequency Spectrum in Figure 1

Function 7: Feature Selection (Suppression)

Input: $F^{p,q}$: The qth spectrum generated by the pth smartphone OS, $1 \leq p \leq P$, $1 \leq q \leq Q$, where P and Q denote the number of different smartphone OSes and the number of traces available for each smartphone OS

Output: $F'^{p,q}$:The qth feature-extracted spectrum of the pth smartphone OS, $1 \leq p \leq P$, $1 \leq q \leq Q$, K: the number of top K frequency components to keep in feature-extracted spectra

foreach *spectrum in the input* **do**

Calculate mean μ_i and standard deviation σ_i of the magnitude of frequency components;

`// without loss of generality, we assume the` i`th trace is being processed;`

T_i=$\mu_i + 6\sigma_i$

;

Set k_i to be the number of frequency components with magnitude larger than T_i;

end

K=$\frac{\sum_{i=1}^{PQ} k_i}{PQ}$;

`// PQ: the number of labeled traces`

foreach *spectrum* $F^{p,q}$ *in the input* **do**

The corresponding feature-extracted spectrum $F'^{p,q}$ is formed by keeping the top K significant frequency component and suppressing the magnitude of the rest frequency components to zero;

end

Function 8: OS Identification (Suppression)

Input: $F'^{p,q}$: Feature extracted spectrum of each labeled trace, F^x: spectrum of the test trace, K: the number of top K frequency components to keep in feature-extracted spectra

Output: OStype: Smartphone OS Type

Form the feature-extracted spectrum F'^x by keeping the top K significant frequency components in F^x and suppressing the magnitude of the rest frequency components to zero;

OStype = Decision($F'^{p,q}$, F'^x);

`// The Decision function is defined in Function 3`

Function 9: Feature Extraction (Hybrid)

Input: $F^{p,q}$: The qth spectrum generated by the pth smartphone OS respectively, $1 \leq p \leq P$, $1 \leq q \leq Q$, where P and Q denote the number of different smartphone OSes and the number of traces available for each smartphone OS respectively

Output: $F'^{p,q}$: The qth feature-extracted spectrum of the pth smartphone OS, $1 \leq p \leq P$, $1 \leq q \leq Q$, $B_{selected}$=$[b_1, b_2, \cdots, b_L]$: band selection vector where the binary bit b_i indicates whether the ith frequency component is selected, K: the number of top K frequency components to keep in feature-extracted spectra

foreach *spectrum in the input* **do**

  Calculate mean μ_i and standard deviation σ_i of the magnitude of frequency components;

  `// without loss of generality, we assume the` i`th trace is being processed`

  T_i=$\mu_i + 6\sigma_i$;

  Set k_i to be the number of frequency components with magnitude larger than T_i;

end

K=$\frac{\sum_{i=1}^{PQ} k_i}{PQ}$;

`// PQ: the number of labeled traces`

foreach *spectrum* $F^{p,q}$ *in the input* **do**

  The corresponding intermediate spectrum $F''^{p,q}$ is formed by keeping the top K significant frequency components and suppressing the magnitude of the rest frequency components to zero;

end

Divide the intermediate spectra $F''^{p,q}$ of each smartphone OS into two sets Set_A and Set_B;

$B_{selected}$=ga(fitfun, Set_a, Set_b);

`// We use ga to represent any genetic algorithm and ga accepts the definition of the fitness function` $fitfun$ `and outputs values of the variables (in our case the vector` $B_{selected}$`) resulting the maximum of the fitness function. The fitness function` $fitfun$ `is defined in Function 5`

foreach *spectrum in the input* **do**

  for $i \leftarrow 1$ **to** L **do**

    if b_i==*1* **then**

      include the ith frequency component in $F^{p,q}$ to the feature-extracted spectrum $F'^{p,q}$;

    end

    `// without loss of generality, we assume the spectrum` $F^{p,q}$ `is being processed`

  end

end

Function 10: OS Identification (Hybrid)

Input: $F'^{p,q}$: Feature extracted spectrum of each labeled trace, F^x: spectrum of the test trace, K: the number of top K frequency components to keep in feature-extracted spectra, $B_{selected} = [b_1, b_2, \cdots, b_L]$: spectrum selection vector

Output: OStype: Smartphone OS Type

Form the intermediate spectrum F''_x by keeping the top K significant frequency component in F^x and suppressing the magnitude of the rest frequency components to zero;

for $i \leftarrow 1$ **to** L **do**

  if b_i=*1* **then**

    Include the ith frequency component of spectrum F''^x into F'^x;

  end

end

OStype = Decision($F'^{p,q}$, F'^x);

`// The Decision function is defined in Function 3`

be the average number of the frequency components with the magnitude larger than the threshold in each trace.

The feature-extracted spectrum is formed by: (1) keeping the top K frequency components and (2) suppress the magnitude of the rest frequency components to zero. The pseudo code of the feature selection step in the suppression algorithm in shown in Function 7 and the corresponding OS identification step is shown in Function 8.

4.6 Hybrid Algorithm

Finally, we describe an algorithm (which we call *hybrid algorithm*) that combines the effectiveness of the spectrum selection algorithm with the efficiency of the suppression algorithm described earlier. We observe that the suppression algorithm keeps the significant frequency components (i.e. large components) for OS identification. Some of the significant frequency components (such as the frequency components caused by round-trip time) are extraneous to the OS operation and are therefore not useful for smartphone OS identification. The hybrid algorithm removes those sig-

nificant frequency components from the spectrum before proceeding to select the characteristic frequency components.

The feature extraction step in the hybrid identification algorithm works as follows: First, as in the suppression algorithm, only the top K significant frequency components in the spectrum $F^{p,q}$ are kept in the intermediate spectrum $F''^{p,q}$. Then the algorithm searches for the best combination of the remaining frequency components with genetic algorithm. The final feature-extracted spectrum $F'^{p,q}$ is formed by applying the best selection on the intermediate spectrum $F''^{p,q}$.The pseudo code of the feature extraction step and the corresponding OS identification step are shown in Function 9 and Function 10 respectively.

4.7 Comparison of the OS Identification Algorithms

In this section we proceed to compare the four OS identification algorithms described above. The comparison is in terms of identification performance and computational complexity.

As discussed above, the spectrum of any smartphone traffic contains both *characteristic frequency components* and *noise frequency components*. To improve the identification performance, we need to remove the noise frequency components as much as possible. Among the proposed algorithms it is to be expected that the spectrum selection algorithm performs best since it largely relies on a search algorithm to identify the most characteristic frequency components. The suppression algorithm filters out the noise frequency components following the heuristic that most insignificant frequency components tend to be noise frequency components. The hybrid algorithm aims to improve the performance of the suppression algorithm by removing the noise frequency components that are significant. The full spectrum algorithm simply removes the DC frequency component so most noise frequency components are left in

Table I: Complexity of the Feature Extraction and Correlation in the Identification Algorithms (P: Number of Different Smartphone OSes, Q: Number of labeled traces available for each smartphone OS, K: Number of most significant frequency components to keep, L: Length of Spectra)

	Full Spectrum	Spectrum Selection	Suppression	Hybrid
Feature Extraction	O(PQ)	Complexity of Genetic Algorithms	O(PQ)	Complexity of Genetic Algorithms
Correlation	O(L)	Dependent on Selection Results	O(K)	O(K)

the spectrum.

The identification performance advantages of the selection algorithm a come at the cost of a significant increase in computational complexity, and the suppression and hybrid algorithm both aim at reducing this cost. The four identification algorithms differ in the feature extraction step and the OS identification step, while the other two steps do not vary. In the following comparison we focus on the computational complexity of these two steps.

A qualitative comparison of the complexity of the feature extraction step during the training phase of the four algorithms is shown in Table I. The feature extraction step in both the spectrum selection and the hybrid algorithm is most time-consuming since it needs to use an optimization step (genetic algorithm in our case) to find the best combination of frequency components for identification. The complexity of feature extraction step in the full spectrum algorithm and the suppression algorithm is much lower since the step in the full spectrum algorithm only needs to remove the DC component from each spectrum and the suppression only needs to suppress insignificant frequency components in each spectrum.

We also want to point out that the cost for feature extraction is only incurred during the training phase. The complexity of the feature extraction step will not affect the cost during operation.

The most time-consuming part of the OS identification step is the correlation. The complexity of correlating two spectra is $O(L)$ where L denotes the length of the spectra. Thus, the complexity of the correlation in the suppression algorithm is much

lower, since a suppressed spectrum can be represented by a sparse vector leading to a correlation cost of $O(K)$ when we keep the K most significant components. The correlation in the hybrid algorithm is of the same complexity because of the suppression in the hybrid algorithm. For the full spectrum algorithm, the complexity of the correlation is O(L) since only the DC component is removed. The complexity of the correlation in the spectrum selection algorithm depends on the selection result and it is usually higher than the suppression and the hybrid algorithms.

CHAPTER V

EMPIRICAL EVALUATION

In this chapter, we evaluate the identification performance of the proposed identification algorithms. The evaluation is based on 336 GB of smartphone traffic collected over more than three months on different smartphone OSes.

5.1 Experiment Setup

The experiment setup is shown in Figure 5. The smartphones with different OSes are used to watch YouTube streaming video, download files with the HTTP protocol, and make video calls with Skype. These three applications are selected for our experiments because of their popularity and their availability on different smartphone OSes: (1) The three applications are among the most popular applications according to the number of downloads shown in application stores. (2) We want to avoid the applications that are only available on one specific smartphone OS. Because if the OS-specific applications are chosen, then OS identification is equivalent to application identification. The three applications are available on all the smartphone

Table II: Specifications of Smartphones Used

Phone	Operating System	CPU	RAM	Internal Storage	Battery
HTC Desire HD	Android v2.3	1 GHz Scorpion	768 MB	1.5 GB	1230 mAh
iPhone 4S	iOS 5	1 GHz Cortex-A9 Dual-Core	512 MB	16 GB	1432 mAh
Nokia Lumia N8	Symbian 3	680 MHz ARM 11	256 MB	16 GB	1200 mAh
Nokia Lumia 900	Windows Phone 7.5	1.4 GHz Scorpion	512 MB	16 GB	1830 mAh

OSes studied in the project. If multitasking is supported in a smartphone OS, we also use the smartphone for video streaming, file downloading, and Skype video calls at the same time. The wireless traffic from the smartphone is collected by an HP dc7800 computer. The data collection is through a Linksys Compact Wireless USB adapter (WUSB54GC) installed on the computer. The wireless access points used in the experiments include both the wireless router in our research lab and wireless access points managed by the university. [1]

The smartphone OSes included in our experiments are Apple's iOS, Google's Android OS, Windows Phone OS, and Nokia Symbian OS. According to the market research by Gartner [6], these four smartphone OSes have taken 94.8% of the market share in 2012 and Android OS is taking 69.7% of the market share. The smartphones used in our experiments are listed in Table II. For each possible combination of the smartphone OS and the application, at least 30 traffic traces of 50 minutes long are collected.

[1] In a different scenario, the data-collecting machine may be monitoring the traffic on the wired portion of the traffic path. The scenario chosen for our experiments is representative of a drive-by or walk-by attack.

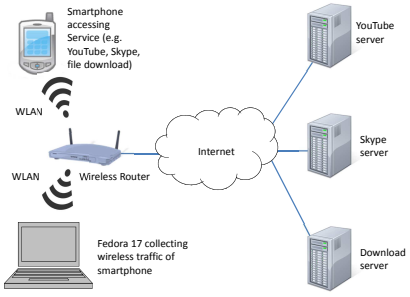


Figure 5: Data Collection Setup

5.2 Performance Metrics

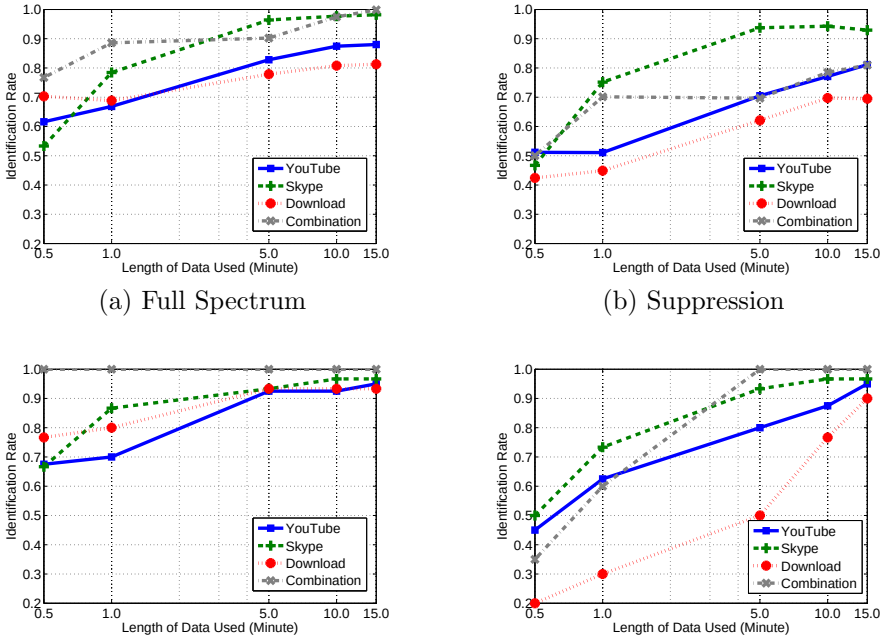
The identification performance is measured with the following three performance metrics: (1) *identification rate* defined as the ratio of successful identifications to the number of attempts, (2) *false negative rate* defined as the proportion of traces generated by smartphone OS, say Y, identified as traces generated by other smartphone OSes, and (3) *false positive rate* defined as the proportion of the traces generated by other smartphone OSes identified as traces generated by smartphone OS Y.

False positive rate and false negative rate can provide more detailed performance information than the identification rate since the false positive rate and false negative rate are specific to each type of smartphone OS. On the other hand the identification rate, averaged across all the smartphone OSes, can show us the overall identification performance.

5.3 Length of Traffic Traces

Our first set of experiments focus on the length of the traffic traces used for OS identification. The traffic used in OS identification includes YouTube video streaming traffic, file downloading traffic, Skype traffic, and combined traffic. The combined traffic is collected by running YouTube video streaming, file downloading, and Skype video calls simultaneously on the OSes supporting multitasking. For brevity, we call the four types of traffic YouTube Traffic, Download Traffic, Skype Traffic, and Combined Traffic respectively in the rest of the paper.

The sample interval used in this set of experiments is of length 8ms. For each type of traffic and each smartphone OS, we collected 30 traces. In this set of experiments, we use 20 of the 30 traces as labeled traces and the remaining 10 traces



(a) Full Spectrum (b) Suppression

(c) Spectrum Selection (d) Hybrid

Figure 6: Identification Performance with Traces of Different Length

as test traces. The experiment results are obtained with 1000 random combinations of the 20 labeled traces and 10 test traces.

The experimental results of the four proposed identification algorithms are shown in Figure 6.

Full Spectrum Identification Algorithm: From Figure 6(a), we can make the following observations: (1) The highest identification rates with YouTube Traffic,

Table III: Application Resource Consumption (The consumption data is obtained by running each application alone for 50 minutes on the devices. Since the Symbian OS and the Windows phone do not have built-in monitoring tools, we do not collect the data for the two OSes.)

Application (OS)	Battery Consumption	CPU Usage
YouTube (iOS)	10%	35%
Skype (iOS)	25%	40%
Download (iOS)	10%	15%
YouTube (Android)	19%	20%
Skype (Android)	50%	74%
Download (Android)	57%	36%

Download Traffic, Skype Traffic, and Combined Traffic are 0.88, 0.98, 0.81, and 0.99 respectively. (2) Even with only 30 second traffic traces, the identification rates are 0.61, 0.53, 0.70, and 0.76 for YouTube traffic, download traffic, Skype traffic, and combined traffic respectively. (3) In general, the identification rates increase with the length of traces. (4) We can observe that the identification rates on combined traffic are close to or slightly better than the identification rate on Skype traffic. The identification rates of combined traffic are higher than the rates of YouTube traffic and download traffic. The reasons are as follows: (a) Combined traffic has more characteristic frequency components for OS identification in comparison with single type of traffic since combined traffic contains other types of traffic. (b) When a smartphone is more heavily loaded because it is running multiple applications, the OS features such as power saving mechanisms are more frequently used. (5) The identification rates for Skype traffic are higher than the rates for the Download and YouTube traffic. The differences are mainly because of the higher resource consumption, in terms of both power consumption and CPU usage, by Skype as shown in Table III. A higher resource consumption leads to a higher chance that power saving features are triggered by the OS, which in turn generates additional characteristic frequency components.

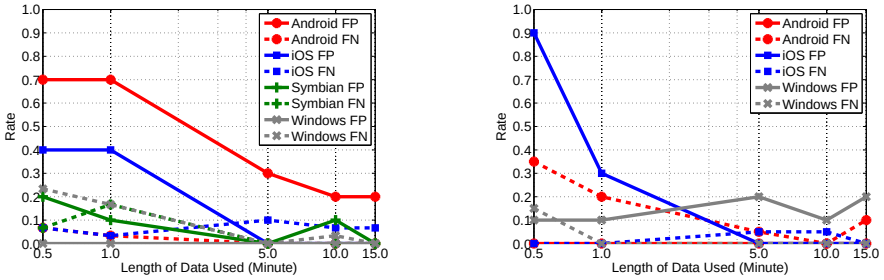
Suppression Identification Algorithm: The parameter K used in the identification step of the suppression algorithm is determined based on the mean and the standard deviation of the magnitude of frequency components as described in Function 7. If not specified, the parameter K will be chosen in the same way in the rest of the paper.

Figure 6(b) shows the experimental results with the suppression algorithm. From Figure 6(b), we can make the following observations: (1) The highest identification rates with YouTube Traffic, Download Traffic, Skype Traffic, and Combined Traffic are 0.81, 0.94, 0.69, and 0.81 respectively. (2) Even with only 30 second traf-

fic traces, the identification rates are 0.51, 0.46, 0.42, and 0.5 for YouTube traffic, download traffic, Skype traffic, and combined traffic respectively. (3) In general, the identification rates increase with the length of traces. (4) We can also have similar observations as in Figure 6(a) that the identification rates of the Skype traffic are higher than the rates of the download and YouTube traffic.

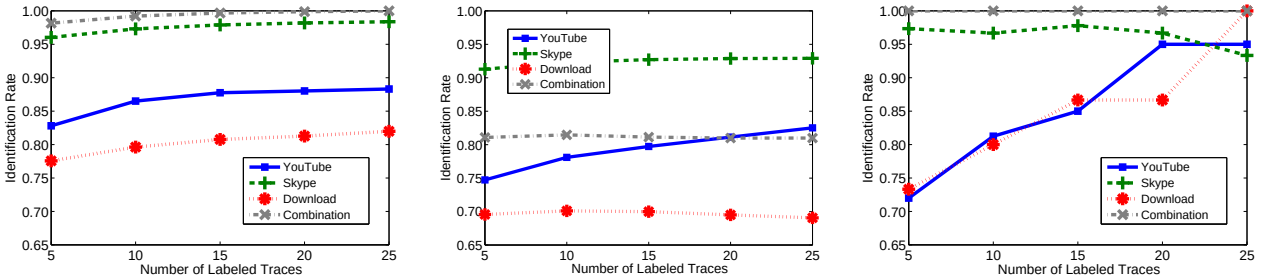
Spectrum Selection Identification Algorithm: Figure 6(c) shows the identification performance of the spectrum selection algorithm. We can observe that the algorithm significantly improves the identification performance: (1) With 30 seconds of traffic traces, the identification can reach .68, .67, .77, and 1 for YouTube traffic, download traffic, Skype traffic, and combined traffic respectively. (2) When comparing with the full spectrum algorithm, the algorithm improves the identification performance by 7%, 0%, 12%, and 1% for YouTube traffic, download traffic, Skype traffic, and combined traffic respectively. When comparing with the suppression algorithm, the improvements are 14%, 2%, 23%, and 19% for YouTube traffic, download traffic, Skype traffic, and combined traffic respectively.

Hybrid Identification Algorithm: The experimental results of the hybrid algorithm are shown in Figure 6(d). In comparison with the suppression algorithm, the hybrid algorithm can achieve higher identification rates as shown in Figure 6(d). The improvement is mainly because of the hybrid algorithm's ability to filter out noise frequency components using a genetic algorithm. But when compared with the spectrum selection algorithm, the identification rates of the hybrid algorithm are lower. The performance differences are because of the assumption made in both the hybrid algorithm and the suppression algorithm: most of the insignificant frequency components are noise frequency components. Removing all the insignificant frequency components means that some characteristic frequency components that are insignificant may also be removed.



(a) YouTube (b) Skype

Figure 7: False Alarm Rates (Spectrum Selection Algorithm)



(a) Full Spectrum (b) Suppression (c) Spectrum Selection

Figure 8: Identification Performance with Different Number of Labeled Traces

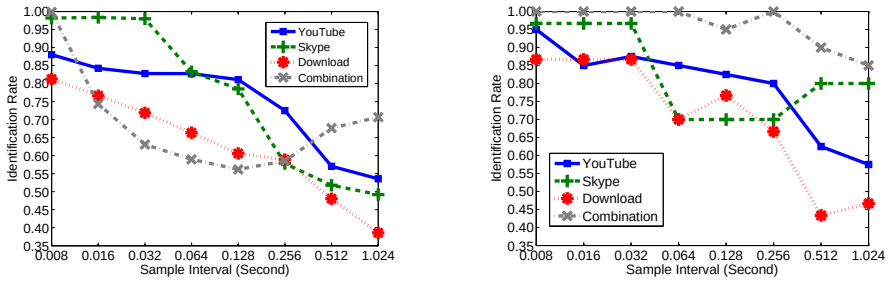
False Alarm Rates

To investigate the identification performance on each OS, we use false alarm rates as the performance metrics. Figure 7 shows the false alarm rates of the OS identification with the spectrum selection algorithm. Similar observations can be made on the other identification algorithms.

From Figure 7, we observe that the false alarm rates decrease with the length of traces. When the trace length approaches 15 minutes, the false alarm rates are below 20%.

5.4 Number of Labeled Traces

The number of labeled traces can affect the identification performance. To evaluate the effect, we vary the number of the labeled traces from five to 25 in this



(a) Full Spectrum (b) Spectrum Selection

Figure 9: Identification Performance with Different Sample Intervals

set of experiments. The rest of the parameters used in the experiments are set as follows: (1) The sample interval is set to be 8ms. (2) The length of traces used in the experiments is 15 minutes.

Figure 8 shows the experimental results with different number of labeled traces. The results show: (1) Even with five labeled traces, the identification of the spectrum selection algorithm can reach 1, 0.97, 0.72, and 0.73 with Combination, Skype, YouTube, and Download Traffic respectively. (2) The identification rate increase with the number of labeled traces. The results are compliant with our expectation since more labeled traces can lead to better knowledge of the characteristic frequency components.

5.5 Sample Interval

One of the critical parameters that will affect the identification performance is the length of the sample interval. The effect of sampling is equivalent to low pass filtering since the fluctuation within a sample interval is completely removed. The sampling can be helpful for OS identification if the sampling filters out noise frequency components. The sampling can also degrade the identification performance if the sampling filters out characteristic frequency components.

We vary the length of sample intervals from 8ms to 1024ms. The identification performance of the full spectrum algorithm and the spectrum selection algorithm is shown in Figure 9(a) and Figure 9(b) respectively. The figures show that the best identification performance is achieved when the sample interval is short like length 8ms for both identification algorithms. We can also observe the decrease of the identification rate when the sample intervals becomes larger. The decrease is simply because a large sample interval means less number of samples available for OS identification. We also observe some fluctuation in both figures. We believe it is because of the filtering effect of the sampling interval: If some characteristic frequency components are filtered out, the identification rates may go down. If some noise frequency components are filtered out due to the sample interval, the identification performance can be improved. But in general, for spectrum selection algorithm, it is better to use a small sample interval so that most frequency components are kept and the spectrum selection algorithm can use a genetic algorithm to remove the noise frequency components.

5.6 Number of Top Significant Frequency Components Kept (Suppression Algorithm)

As discussed in Section 4.5, the key parameter of the suppression algorithm is K, the number of most significant frequency components to keep for identification. In the suppression algorithm, the parameter K is determined in the feature selection step (Function 7) based on the mean and standard deviation of the magnitude of frequency components in the spectra of the labeled traces. The parameter K calculated in the feature selection step is then used in the OS identification step (Function 8) to identify smartphone OSes.

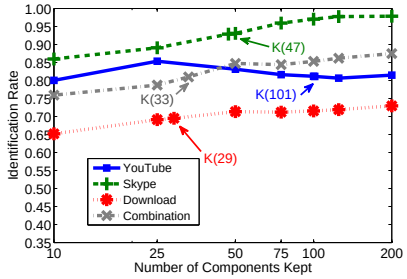


Figure 10: Number of Top Significant Frequency Components to Keep in the Suppression Algorithm. K is determined in the feature selection step (Algorithm 7) based on the mean and standard deviation of the magnitude of frequency components in the spectra of the labeled traces.

In this set of experiments, we vary the parameter K used in the OS identification step of the suppression algorithm. The value of K calculated based on the mean and standard deviation is also included in the experiments.

Figure 10 shows the identification performance with different values for K. We observe: (1) When the number of frequency components to keep is relatively small, the identification rate is relatively low. It is because lots of significant frequency components that can be helpful for identification are not selected. (2) Similarly, when K becomes too large, the identification rate may drop as well. This is expected as the number of noise frequency components that are considered as features grows. (3) We also observe that when the number of frequency components to keep is calculated based on the mean and the standard deviation as in Algorithm 7, the identification rate is close to the maximum for YouTube and download traffic. For Skype and combination traffic, the identification rates increase slightly after the number of frequency components calculated based on the mean and standard deviation. We gather that Function 7 effectively selects a set of frequency components to be retained and so filters out the noise frequency components and keeps the characteristic frequency components.

In comparison with the identification performance of other algorithms shown in Figure 6(a) and Figure 6(c), the identification performance of the suppression algorithm is very close to the performance of the full spectrum algorithm and worse than the performance of the spectrum selection algorithm. The performance differences indicate that some characteristic frequency components may not be among the most significant frequency components. But the heuristic that most characteristic frequency components are significant frequency components works well since (1) the suppression algorithm can greatly reduce the computational cost as described in Section 4.7 in comparison with the full spectrum algorithm, and (2) the performance of the suppression algorithm is similar to the full spectrum algorithm and comparable to the performance of the spectrum selection algorithm.

CHAPTER VI

DISCUSSION

6.1 Application Identification

The algorithms proposed above for OS identification can be naturally extended to identify applications running on a target device. Obviously, the application information can be critical to an adversary planning security and privacy attacks to the target device.

We use the spectrum selection algorithm to briefly illustrate how to apply the techniques described earlier to the identification of applications. The algorithm itself does not need to be modified in any way other than being trained on application-generated traces rather than OS generated ones.

We apply the spectrum selection algorithm to detect applications using the same traffic traces described in Chapter 5. The applications to be detected include YouTube, Skype, and Downloading. Figure 11 shows the performance of the application detection along with the false-positive and false-negative identification rates for each application included in the experiment. According to Figure 11, the application

Table IV: Empirical running times for the Full Spectrum, Suppression, Spectrum Selection and Hybrid OS detection algorithms. 15 minutes of traffic with an 8 ms sample interval is used to generate this data.

	Training (Minute)				Identification (Second)			
	YouTube	Skype	Download	Combination	YouTube	Skype	Download	Combination
Full Spectrum	0	0	0	0	0.2636	0.1389	0.1397	0.0622
Suppression	0.0075	0.0048	0.0024	0.0014	0.2930	0.1715	0.1707	0.0820
Spectrum Selection	759.6	493.2	336.8	161.4	0.1390	0.0814	0.0836	0.0435
Hybrid	3396	529.3	1727	541.3	0.1118	0.0639	0.0817	0.0299

detection rate can reach 85% with 15 minutes of traffic traces. The results show that the spectrum selection algorithm can successfully extract application-specific frequency components for application detection.

6.2 Running Time

Table IV shows the empirical running time for training and identification for all four algorithms. First it can be observed that each identification for all algorithms takes less than 0.3 seconds. It means the attack is very efficient and feasible once the training is finished. We can also observe that the time required for training varies due to the complexity of the algorithm. The spectrum selection and hybrid algorithms take more time for training because of the genetic algorithm used in the two algorithms. The time required for training with the full spectrum algorithm is

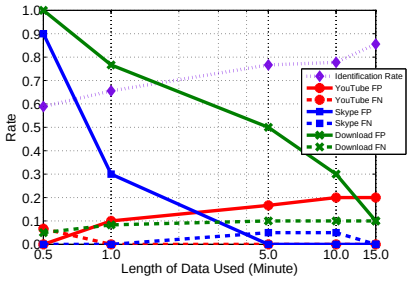


Figure 11: Application Identification Performance (Spectrum Selection). The applications to be detected are YouTube, Skype, and Downloading. The applications are equally distributed making the random guess rate 33%.

zero since no training is required. The full spectrum algorithm just removes the DC component and uses the rest as *characteristic frequency components*.

CHAPTER VII

CONCLUSION

In this paper we present, evaluate, and compare a number of frequency-domain analysis based algorithms in order to illustrate how susceptible smartphones are against passive attacks that aim at inferring configuration parameters of a target smartphone despite the phone using encrypted communication. Specifically, we show how it is possible, based on relatively short measurements, to infer the OS of the phone or the particular application that is running.

Our future work will focus on the countermeasures. One can envision a number of possible countermeasures to this type of attack, some of which have been previously used with varying success in other domains. For example, the OS can attempt to mask its timing footprint by disturbing the timing of outgoing packets. This has to be implemented very carefully, as previous results of traffic padding and batching in the area of anonymous communication have shown. Perturbing the timing behavior of outgoing packets has a number of effects on applications and interactions across the network. For latency sensitive applications, for example, any perturbing of the timing leads to a reduction of timing slack, which makes it the more difficult to satisfy

end-to-end latency requirements. Similarly, it has been shown that perturbances of timings on TCP packets can cause visible secondary timing footprints caused by end-to-end TCP dynamics and so may be counter-productive.

A typical method to counter application detection based on traffic flows is to aggregate the flows from multiple applications into a single flow. As a result, the timing characteristics of the individual applications become much more difficult to detect. In our previous work on anonymous communication we have been applying blind-source separation techniques to separate traffic components and attribute them to senders. In our current and future work, we are investigating how similar mechanisms can be applied to render OS and application identification effective despite flow aggregations.

BIBLIOGRAPHY

[1] A. Biryukov, I. Pustogarov, and R.-P. Weinmann. Trawling for tor hidden services: Detection, measurement, deanonymization. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, May 2013.

[2] X. Cai, X. Zhang, B. Joshi, and R. Johnson. Touching from a distance: Website fingerprinting attacks and defenses. In *Proceedings of the 19th ACM conference on Computer and Communications Security (CCS 2012)*, October 2012.

[3] M. Charts. Wifi mobile phone traffic grows. `http://www.marketingcharts.com/wp/direct/wifi-mobile-phone-traffic-grows-19604/`, October 2011.

[4] X. Chen, R. Jin, K. Suh, B. Wang, and W. Wei. Network performance of smart mobile handhelds in a university campus wifi network. In *Proceedings of the 2012 ACM conference on Internet measurement conference*, IMC '12, pages 315–328, New York, NY, USA, 2012. ACM.

[5] Z. Durumeric, E. Wustrow, and J. A. Halderman. Zmap. `https://zmap.io/`.

[6] Gartner. Gartner says worldwide mobile phone sales declined 1.7 percent in 2012. `http://www.gartner.com/newsroom/id/2335616`, February 2013.

[7] D. Gayle. This is a secure line: The groundbreaking encryption app that will scramble your calls and messages. `http://www.dailymail.co.uk/sciencetech/article-2274597/How-foil-eavesdroppers-The-smartphone-encryption-app-promises-make-communications-private-again.html`, February 2013.

[8] X. Gong, N. Borisov, N. Kiyavash, and N. Schear. Website detection using remote traffic analysis. In *Proceedings of the 12th Privacy Enhancing Technologies Symposium (PETS 2012)*. Springer, July 2012.

[9] L. Greenemeier. Cloud warriors: U.s. army intelligence to arm field ops with hardened network and smartphones. `http://www.scientificamerican.com/article.cfm?id=us-army-intelligence-cloud-smartphone`, March 2013.

[10] S. Grimes. App to provide military-level encryption for smartphones. `http://www.ksl.com/?nid=1014\&sid=22513938`, October 2012.

[11] D. Herrmann, R. Wendolsky, and H. Federrath. Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier. In *Proceedings of the 2009 ACM workshop on Cloud computing security (CCSW '09)*, pages 31–42, New York, NY, USA, October 2009. ACM.

[12] J. Huang, Q. Xu, B. Tiwana, Z. M. Mao, M. Zhang, and P. Bahl. Anatomizing application performance differences on smartphones. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, MobiSys '10, pages 165–178, New York, NY, USA, 2010. ACM.

[13] E. Kollmann. Chatter on the wire: A look at extensive network traffic and what it can mean to network security. `http://chatteronthewire.org/download/OS%20Fingerprint.pdf`, August 2005.

[14] M. Liberatore and B. N. Levine. Inferring the Source of Encrypted HTTP Connections. In *Proceedings of the 13th ACM conference on Computer and Communications Security (CCS 2006)*, pages 255–263, November 2006.

[15] S. J. Murdoch. Hot or not: Revealing hidden services by their clock skew. In *Proceedings of CCS 2006*, November 2006.

[16] Netresec.com. Passive os fingerprinting. `http://www.netresec.com/?page=Blog&month=2011-11&post=Passive-OS-Fingerprinting`, November 2011.

[17] Nmap.org. Nmap network scanning. `http://nmap.org/book/osdetect.html`.

[18] A. V. Oppenheim, A. S. Willsky, and S. H. Nawab. *Signals & Systems (2nd ed.)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

[19] L. Øverlier and P. Syverson. Locating hidden servers. In *Proceedings of the 2006 IEEE Symposium on Security and Privacy*. IEEE CS, May 2006.

[20] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel. Website fingerprinting in onion routing based anonymization networks. In *Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2011)*. ACM, October 2011.

[21] H. Project. Know your enemy: Passive fingerprinting. `http://old.honeynet.org/papers/finger/`, March 2002.

[22] M. Smart, G. R. Malan, and F. Jahanian. Defeating tcp/ip stack fingerprinting. In *Proceedings of the 9th conference on USENIX Security Symposium - Volume 9*, SSYM'00, pages 17–17, Berkeley, CA, USA, 2000. USENIX Association.

[23] G. Tzagkarakis, M. Papadopouli, and P. Tsakalides. Singular spectrum analysis of traffic workload in a large-scale wireless lan. In *Proceedings of the 10th ACM Symposium on Modeling, analysis, and simulation of wireless and mobile systems*, MSWiM '07, pages 99–108, New York, NY, USA, 2007. ACM.

[24] S. Zander and S. J. Murdoch. An improved clock-skew measurement technique for revealing hidden services. In *Proceedings of the 17th USENIX Security Symposium*, July 2008.

[25] F. Zhang, W. He, X. Liu, and P. 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.