This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2019.2892482, IEEE Transactions on Dependable and Secure Computing

Towards Smartphone Operating System Identification

Ye Zhu, Nicholas Ruffing, Jonathan Gurary, Yong Guan, and Riccardo Bettati

***Abstract*—Smartphone reconnaissance, the first step to launch security attacks to a target smartphone, enables an adversary to tailor attacks by exploiting the known vulnerabilities of the target system. In this paper we investigate smartphone OS identification with encrypted traffic. We propose four algorithms to do that, which are based on the spectral analysis of the encrypted traffic. The 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 experiment results show that the algorithms can identify the smartphone OS accurately. The identification accuracy can reach 100% with only 30 seconds of smartphone traffic.**

I. Introduction

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 or 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 also to share sensitive data, such as personal contacts and banking information. Smartphones are also adopted in business and military environments [14] 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. 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. Examples include data exfiltration, such as [7] for Android devices, and [8] for non-compute devices. Other platform-specific attacks, primarily targeting iOS devices, are described in [9], [10], [11]. The most readily obtainable information that enables OS identification is the wireless traffic generated by the target smartphone.

When the traffic is encrypted, the eavesdropper is naturally prevented from accessing packet content. In some situations, such as SSL/TLS encrypted traffic [6], packet header information may remain observable, and may continue to provide useful information to the eavesdropper. In other situations, such eavesdropping on encrypted IEEE 802.11 links [32] or on encrypted LTE links [1], the eavesdropper's ability to monitor the traffic is limited to the timing of the packets and – in the absence of packet padding – packet sizes. In this paper we will focus on packet timing: Observations indicate that timing side channels can be leveraged effectively, since different OSes cause the smartphone to generate traffic with different timings. Such 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.

In this paper we describe how differences in OSes can be identified by analyzing of the timing traces of the generated traffic in the frequency domain. Frequency domain analysis is a classical tool to analyze temporal signals [22] by converting signals from the time domain to the frequency domain.

The main challenge in OS identification with 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, which often are 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

Ye Zhu, Nicholas Ruffing, and Jonathan Gurary are with the Department of Electrical Engineering and Computer Science, Cleveland State University, Cleveland OH 44115, USA. (e-mail: y.zhu61@csuohio.edu, n.ruffing@vikes.csuohio.edu, and j.gurary@vikes.csuohio.edu)

Yong Guan is with the Department of Electrical and Computer Engineering Iowa State University, Ames, IA 50011. (e-mail: yguan@iastate.edu)

Riccardo Bettati is with Department of Computer Science, Texas A&M University, College Station, TX 77843. (e-mail: bettati@cs.tamu.edu)

Manuscript received 2017

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2019.2892482, IEEE Transactions on Dependable and Secure Computing

is $O(L)$ where L denotes the length of the spectra. In this paper, we propose approaches to significantly reduce the computational complexity while continuing to accurately identify smartphone OSes.

Our major contributions are summarized as follows: (1) We propose four smartphone OS identification algorithms, which use frequency spectrum analysis to capture the differences in smartphone OSes. Correlation is used to match the spectrum of interest to the spectra generated by known smartphone OSes. (2) We propose OS identifications algorithms that can remove noise frequency components to improve the identification accuracy. (3) We evaluated the OS identification algorithms with extensive empirical experiments. Our experiments show that the OS identification algorithm can identify smartphone OS with very high accuracy with only small amounts of smartphone traffic. We also try the identification algorithm on minor versions of smartphone OSes. We find the identification performance varies with traffic type. We observer that the identification of minor versions of smartphone OSes is more effective at higher resource consumption levels. (4) 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 experiment results show that high identification accuracy can be achieved with as little as 30 seconds of smartphone traffic.

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

II. Related Work

We investigated a simple OS identification algorithm, the so-called *spectrum selection* algorithm in [27]. In this paper we present and evaluate a family of identification algorithm, which allows to trade-off accuracy *vs.* cost: (1) The *suppression algorithm* is designed to significantly reduce the cost of selecting characteristic frequency components in the spectrum selection identification algorithm. This algorithm reduces the cost through a heuristic approach. (2) The *hybrid algorithm* combines the effectiveness of the original spectrum selection algorithm with the efficiency of the suppression algorithm. This algorithm can remove significant frequency components that are extraneous to OS operation. (3) The *full spectrum algorithm* only removes the DC frequency component and leave the remaining frequency components as characteristic frequency components. This algorithm serves as a baseline for comparison. We compare the four identification algorithms in terms of computational complexity and identification performance through extensive experiments.

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

A. OS and Smartphone 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 so 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 [20] 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 [20]. Kollmann [17] proposes to fingerprint the OS based on its implementation of the DHCP protocol, as different OSes support different combinations of DHCP options. The network analysis tools `siphone` and `p0f` developed as a part of the Honeynet Project [25] fingerprint the computer OS by checking four TCP signatures. Two of them are the TTL and the TCP window size as discussed in [20]. 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 [21], 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.* [12], called Zmap, allows a single computer with a gigabit Ethernet connection to scan the entire IPv4 address space in 45 minutes.

Countermeasures have been proposed in order to defeat OS fingerprinting. Smart *et al.* [28] 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 are intended to 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.

Stöber *et al.* [29] describe an approach to identify smartphones by eavesdropping on 3G/UMTS transmissions, which are protected by link-level encryption. As a result, the eavesdropper has access only to arrival time, size, and direction (incoming/outgoing) of packets. By analyzing packet inter-arrival times, packet lengths, and burst lengths, the authors are able to identify the collection of applications (the application mix, including applications that generate background

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2019.2892482, IEEE Transactions on Dependable and Secure Computing

traffic) that are active on the phone. This application mix in turn is shown to be an effective feature for the identification of the specific smartphone. We note that this work has a different attacker model from ours: In [29] the eavesdropper has access to the first-hop 3G/UMTS link, and therefore to the entirety of the traffic sent and received by the device, and she can therefore infer the application mix. In our work, the eavesdropper has only remote access to the device. As a result, the eavesdropper sees the traffic of a single application only, or in some cases a very small set of applications. Trying to remotely infer the application mix is therefore impossible, and we therefore must focus on features belonging to the traffic of a single application.

B. 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.

1) Website Fingerprinting: Herrmann *et al.* [15] 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.* [24] 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%. Camouflaging as a countermeasure to hamper the fingerprinting attack was proposed in [24], and the countermeasure is able to decrease the accuracy to as low as 3%. A website-detection attack that can be executed from a remote location was proposed in [13]. The attack first estimates the load inside a victim's router queue by measuring the round-trip time of regularly-spaced probe packets. Based on this estimation of the load, any of the website fingerprinting methods described above can be used. Cai *et al.* [3] attempted to defeat countermeasures proposed to 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 [18] 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) Inferring Users' Online Activities Through Traffic Analysis: Zhang *et al.* [34] 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 were able to infer the users activities with 80% accuracy using 5 seconds of traffic and 90% accuracy with 1 minute of traffic. Similarly, Conti *et al.* [6] describe an approach to identify user activities by monitoring encrypted traffic of Android devices. The authors assume SSL/TLS protected traffic, which gives the eavesdropper access to information in the packet headers. They leverage this information, together with packet timing and other patterns, to infer detailed user activities, such as sending or receiving email, or accessing a profile on a social network. The authors compare their approaches to [15], [18]. The authors apply the insights from this work to the identification of apps in [30]. Relaxing the need to identify apps rather than user actions allows the authors to automate the fingerprinting as part of a tool. Wang *et al.* [32] are able to identify apps using encrypted IEEE 802.11 traffic. The side channel for this traffic is less rich than is the case SSL/TLS traffic, as packet headers (and therefore sender and receiver IP addresses) are encrypted as well.

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 [23] 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 similar effort in [2] 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 [33] aim to improve their clock-skew measurement technique for revealing hidden services. Their original method [19] correlates clock-skew changes during time of high load.

C. Analysis of Smartphone Traffic

Smartphone traffic has been analyzed for various purposes. In [31] Tzagkarakis *et al.* proposed to use the Singular Spectrum Analysis to characterize network load in a large WLAN. Their findings can help design large-scale WLAN's that can be used by smartphones in large public areas. Chen *et al.* [5] 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 to gather interesting insights on TCP and application behavior of smartphones and their effect on performance. Huang *et al.* [16] proposed a methodology for comparing application performance based on 3G communications. Their study shows how YouTube buffering techniques vary across smartphone OSes.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2019.2892482, IEEE Transactions on Dependable and Secure Computing

Throughout these experiments, the observer has access to header and payload data. In comparison, our work is focusing on the analysis of encrypted traffic.

D. Spectrum Analysis for Traffic Analysis

Frequency analysis has been used in various ways in traffic analysis. Rocha *et al.*, for example, use scalograms in [26] to generate users' web application use profiles using time and frequency components of collected traffic. Zhu *et al.* use frequency-domain correlation techniques in [36] to match ingress and egress traffic in anonymous communication networks. Similarly, Zhu and Bettati use frequency spectrum matching in [35] to generate end-to-end paths from blind-source separated components of traffic in anonymous networks. The frequency spectrum has been used in previous research for various purposes. However, in this paper, we find out that a frequency spectrum consists of frequency components generated by various factors such as OS differences, network dynamics, application content, and user behavior. According to our knowledge, this is the first attempt to extract frequency components that can be used for OS identification with genetic algorithms. The success of the OS identification algorithms opens a new paradigm to extract frequency components for identifying specific factors such as application content. The success of the component extraction enables traffic analysis to shift from the original flow-level analysis to component-level analysis. One more difference is that previous research is based on frequency spectrum of packet-count time series and the spectrum generated by this paper is based on byte-count time series.

III. Network and Threat Model

A. Network Model

In this paper, our goal is to identify the operating system of a target smartphone that communicates over a wireless link, be it WiFi or 4G/LTE. In both cases, we assume that the communication is encrypted at link layer.

The rationale for OS identification is as follows: (1) On one hand, to launch an attack, the 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/UMTS, or even 4G/LTE, nearly every smartphone on the market is capable of WiFi communication. Next, the majority of traffic from smartphones is sent through WiFi [4] partly because of its low cost and relatively high bandwidth. Finally, WiFi based passive attacks are easy to stage, for example as drive-by or walk-by attacks.

In this paper we focus on the case of WiFi traffic that has been encrypted as link level, for example through WAP or similar mechanisms. Since link-level encryption for WiFi traffic encrypts the entire exchanged frame, the attacker has no access to packet-header information, and has to rely on timing and packet-size information only.

B. Threat Model

In this paper we assume a passive adversary who is able to capture wireless packets exchanged by the target smartphone. The smartphone, in turn, communicates over an encrypted wireless link.

We assume that the adversary has the following capabilities: (1) The adversary is able to eavesdrop on WiFi communications from the target smartphones and collect encrypted traffic for the identification. (2) The WiFi communications are encrypted at link level, for example through the use of WAP. The adversary has therefore no access to packet-header information. (3) The adversary is able to collect traffic from known smartphone OSes and analyze the traffic for future identification. (4) We assume a passive adversary. That is, the adversary is not allowed to add, delete, delay, or modify existing traffic for OS identification. (5) The traffic traces, including the traffic traces collected for training on known smartphone OSes and the traffic traces of interest for identification by the adversary, 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.

IV. Identifying Smartphone Operating Systems

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 policies, and different implementations of the TCP/IP protocol stack. These differences in the OS implementations in turn 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, memory, and battery lifetime. To better utilize these resources, smartphone OSes adopt a number of policies for resource control, including power-management policies.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2019.2892482, IEEE Transactions on Dependable and Secure Computing

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 will very likely generate network traffic differently. Another example is YouTube: In [16], iPhone is reported to first download a portion of video at a high rate, pause for 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 [16] 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 obviously give rise to different timing behaviors for the traffic generated by different smartphone OSes. These differences can be easily captured in the frequency domain. A typical spectrum of YouTube video streaming on Android OS is shown in Figure 1: We 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). The periodic nature of the buffering now becomes evident. 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). We 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 performance.

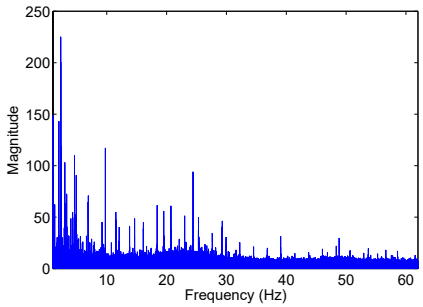
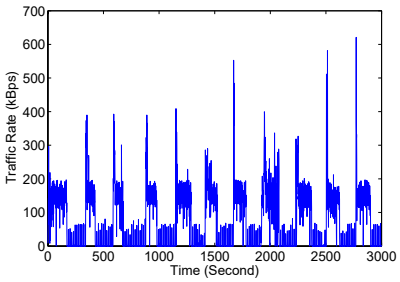
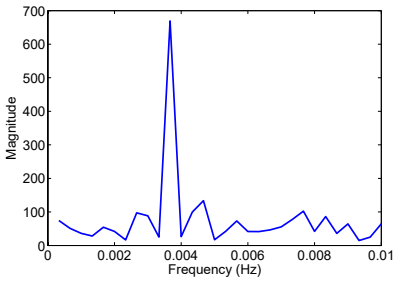


Fig. 1. A Sample Frequency Spectrum of YouTube Streaming Traffic on Android OS (To generate the spectrum, 50 minutes of streaming traffic with an 8 ms sample interval is used.)



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



(b) Zoomed-in part of the spectrum from Figure 1

Fig. 2. Correspondence between Periodicities in a Time Domain Signal and Characteristic Frequency Components

A. Identification Framework

We propose four identification algorithms for OS identification. The four algorithms are designed under the same framework. So before introducing the details of each algorithms, we present the framework first.

The identification can be divided into two phases : *training phase* and *identification phase*. The training phase in turn consists of two steps: *spectrum generation* and *feature extraction*. The identification phase consists of two steps as well, namely the *spectrum generation* and *OS identification*. We describe the details of each step below.

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 the number of bytes received during the ith sample interval divided by the length T of the 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:

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

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:

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

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2019.2892482, IEEE Transactions on Dependable and Secure Computing

$$f_k^S = |y_k|, k = [1, 2, \cdots, M] \quad (2)$$

where the operator $|\cdot|$ denotes the absolute value. Because of the symmetry of the spectrum [22], 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.

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.

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 section.

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:

$$corr(F'^x, F^{A_p}) = \frac{\sum_{k=1}^{L} (f_k'^x - \overline{F'^x})(f_k'^{A_p} - \overline{F'^{A_p}})}{\sqrt{\sum_{k=1}^{L} (f_k'^x - \overline{F'^x})^2 \sum_{k=1}^{L} (f_k'^{A_p} - \overline{F'^{A_p}})^2}}, \quad (3)$$

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 the 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.

B. Full Spectrum

The full spectrum identification algorithm includes all frequency components except for the DC component as feature frequency components. The DC component (also called "zero-frequency component") is the first frequency in the Fourier Transform of a signal, and it represents the average signal value, which in our case is the average traffic rate. We remove the DC component because the average traffic rate largely depends on the content of the traffic (and therefore on the application) rather than on the smartphone OS.

The spectrum generation step of the full spectrum identification algorithm is the same as the step described in Section IV-A. The pseudo code of the feature extraction step and the OS identification step can be found in Function 1 and Function 2, respectively.

Function 1: Feature Extraction (Full Spectrum)

Input: $F^{p,q}$: The qth spectrum generated by the pth smartphone OS, $1 \le p \le P$, $1 \le q \le Q$, 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, $1 \le p \le P$, $1 \le q \le Q$

```
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
```

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 ← 1 to L − 1 do
    f'^x_k = f^x_{k+1};
end
OStype = Decision(F'^{p,q}, F'^x);
```

C. 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. These include *characteristic frequency components*,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2019.2892482, IEEE Transactions on Dependable and Secure Computing

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;

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. Note that there may be frequency components that are caused by OS activities that are very similar across different OSes. Since these frequency components are not helpful for OS identification, one can just as well treat them as noise components without affecting the performance of the identification. We will do just that in this paper.

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 the possible combinations. To make this approach practical, we apply a genetic algorithm to decide which frequency component 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, each of which indicating 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 for the feature extraction step in the spectrum selection algorithm is shown in Function 4 and that for the OS identification step is shown in Function 6. The fitness function used in Function 4 is shown in Function 5.

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
$Rate_{Identification} = \frac{success}{number\ of\ traces\ in\ Set_B}$;

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2019.2892482, IEEE Transactions on Dependable and Secure Computing

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 function is defined in Function 3

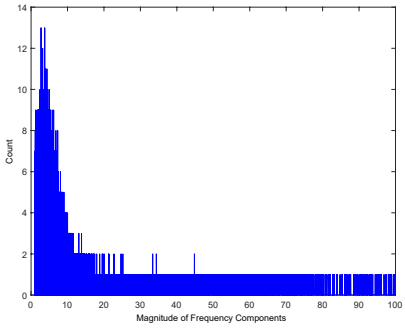


Fig. 3. Magnitude Distribution of the Frequency Spectrum in Figure 1

D. 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 frequency 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 must be about 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 3. 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 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) suppressing 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.

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 ith 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 function is defined in Function 3

E. 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

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2019.2892482, IEEE Transactions on Dependable and Secure Computing

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 significant 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 a 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.

F. Comparison of the OS Identification Algorithms

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 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 in both the spectrum selection and the hybrid algorithm is most time-consuming since it needs to use an optimization scheme (genetic algorithm in our case) to find the best combination of frequency components for the identification. The complexity of feature extraction in the full spectrum algorithm and the suppression algorithm is much lower: In the full spectrum algorithm it only needs to remove the DC component from each spectrum, and the suppression only needs to suppress insignificant frequency components in each spectrum.

The cost of feature extraction is only incurred during the training phase and therefore will not affect the cost during

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 ith 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 function is defined in Function 3

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2019.2892482, IEEE Transactions on Dependable and Secure Computing

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)

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 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.

V. EMPIRICAL EVALUATION

In this section, we evaluate the identification performance of the proposed identification algorithms. The evaluation is based on 489GB of smartphone traffic collected over more than three months on different smartphone OSes.

A. Experiment Setup

The data collection experiment consists of a number of smartphones, running different OSes, and a data-collection point.

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. (We note that if we were to choose OS-specific applications, then OS identification would equivalent to application identification.) The three applications are available on all the smartphone 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 data collection is through a Linksys Compact Wireless USB adapter (WUSB54GC) installed on a computer, which in turn collects packets using `tcpdump`. The wireless access points used in the experiments include both the wireless router in our research lab and various wireless access points managed by the university. [2]

TABLE II
SPECIFICATIONS OF SMARTPHONES USED

Phone	OS	CPU	RAM
HTC Desire HD	Android v2.3	1 GHz Scorpion	768 MB
Galaxy S4	Android v4.4	1.6 GHz 4-core Cortex-A15	2 GB
iPhone 4S	iOS 5	1 GHz 2-Core Cortex-A9	512 MB
iPhone 5S	iOS 8	1.3 GHz 2-core Cyclone	1GB
Nokia Lumia N8	Symbian 3	680 MHz ARM 11	256 MB
Nokia Lumia 900	Win Phone 7.5	1.4 GHz Scorpion	512 MB

The smartphone OSes included in our experiments are Apple's iOS, Google's Android OS, Windows Phone OS, and Nokia Symbian OS. For each possible combination of the smartphone OS and the application, at least 30 traffic traces of 50 minutes each are collected.

B. 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.

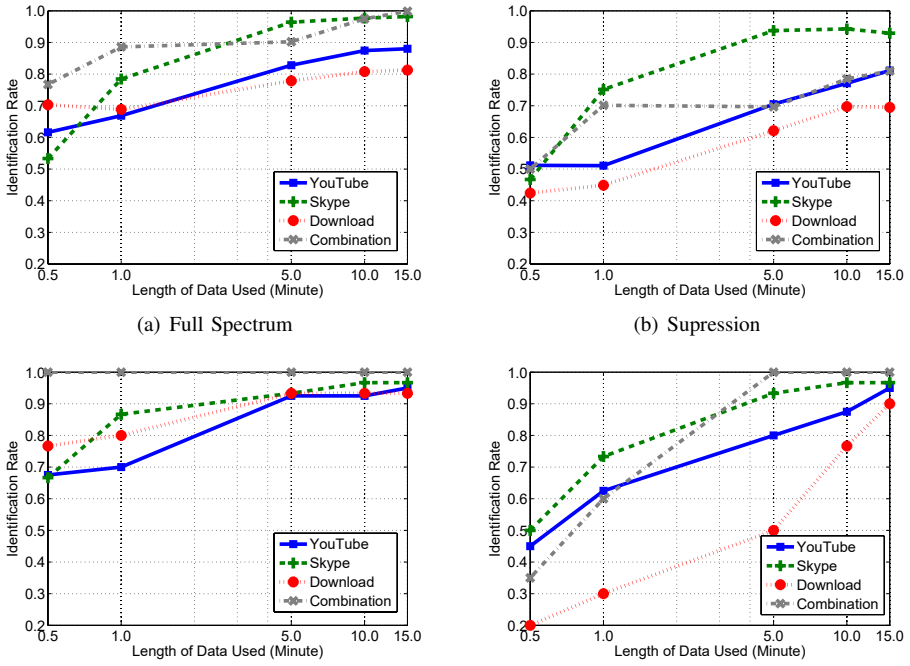
False positive rate and false negative rate can provide more detailed performance information than the identification rate since the false positive rate and false negative rate are specific to each type of smartphone OS. On the other hand the identification rate, averaged across all the smartphone OSes,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

C. Length of Traffic Traces

Our first set of experiments focus on the length of the traffic traces used for the OS identification. The traffic used in the 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 type of traffic as YouTube Traffic, Download Traffic, Skype Traffic, and Combined Traffic respectively in the rest of the paper.

[2] 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.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2019.2892482, IEEE Transactions on Dependable and Secure Computing



(a) Full Spectrum (b) Supression

(c) Spectrum Selection (d) Hybrid

Fig. 4. Identification Performance with Traces of Different Lengths

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%

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 rest 10 traces as test traces. Since the number of possible combinations of choosing 20 traces out of 30 traces as labeled traces is huge, we randomly pick 1000 combinations. The experimental results for the four proposed identification algorithms are obtained with these 1000 random combinations and are shown in Figure 4.

Full Spectrum Identification Algorithm: From Figure 4(a), we can make the following observations: (1) The highest identification rates with YouTube Traffic, 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 4(b) shows the experiment results with the suppression algorithm. From Figure 4(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 traffic traces, the identification

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2019.2892482, IEEE Transactions on Dependable and Secure Computing

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 4(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 4(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 experiment results of the hybrid algorithm are shown in Figure 4(d). In comparison with the suppression algorithm, the hybrid algorithm can achieve higher identification rates. The improvement is due mainly to fact that the hybrid algorithm filters out noise frequency components by genetic algorithms. When compared with the spectrum selection algorithm, the identification rates of the hybrid algorithm are lower, however. 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 be removed as well.

D. False Alarm Rates

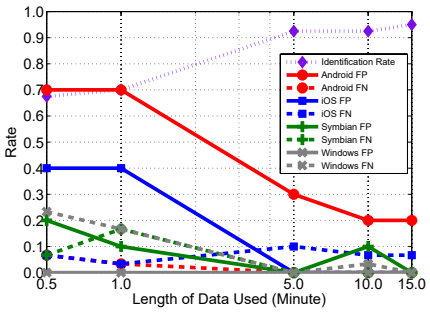
To investigate the identification performance on each OS, we use false alarm rates as the performance metrics. Figure 5 shows the false alarm rates of the OS identification with the spectrum selection algorithm. Similar observations can be made for the other identification algorithms.

We also observe that the false alarm rates decrease with the length of traces. When the trace length approaches 15 minutes, the false alarm rates fall below 20%.

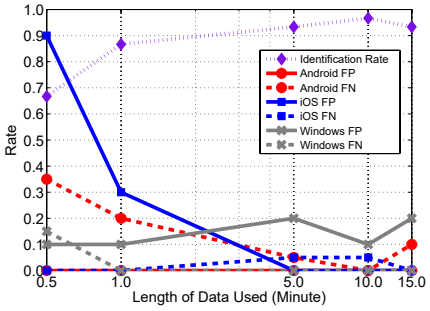
E. Identification of Versions of Smartphone OSes

In this set of experiments, we investigate the possibility of identifying smartphone OS versions running on the same smartphones. We collect traffic from iPhone 5s running iOS 8.0 and iOS 11.4 and iPhone 6s Plus running iOS 11.03 and iOS 11.4. The identification results of both phones are shown in Figure 6.

In Figure 6(a), we can observe that the identification rate can reach 100% for Skype, YouTube, and download traffic. For the combined traffic, the identification rate can reach 83% with 15-minute-long traces. The results show that the identification algorithm can successfully identify the OSes running on iPhone 5s.

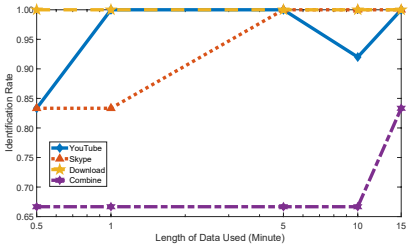


(a) YouTube

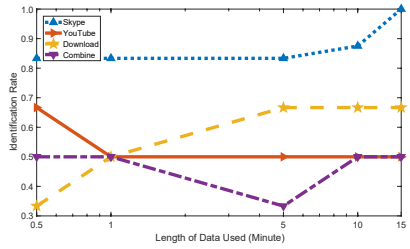


(b) Skype

Fig. 5. False Alarm Rates (Spectrum Selection Algorithm)



(a) iPhone 5S (iOS 8.0 vs. iOS 11.4)



(b) iPhone 6s Plus (iOS 11.03 vs. iOS 11.4)

Fig. 6. OS Version Identification (Spectrum Selection Algorithm)

In Figure 6(b), we can observe the identification rate can achieve 100% for 15-minute-long Skype traffic. But for the other three types of traffic, the identification rates fluctuate around 50%, the random guess rate. When comparing with the results in Figure 6(a) on iPhone 5s, we can observe that the identification performance on iPhone 6s Plus is worse. We believe the reason is that the OS difference between iOS

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2019.2892482, IEEE Transactions on Dependable and Secure Computing

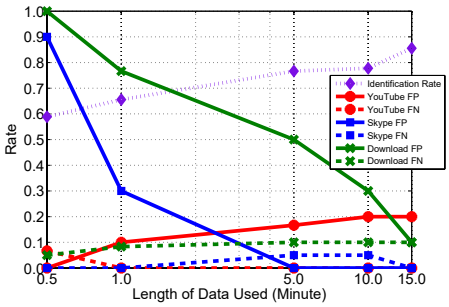


Fig. 7. Application Identification Performance by the Spectrum Selection Algorithm (The applications to be detected are YouTube, Skype, and Downloading. The applications are equally distributed making the random guess rate 33%.)

11.03 and iOS 11.4 is less than the OS difference between iOS 8.0 and iOS 11.4. But the OS difference between iOS 11.03 and iOS 11.4 can still be detected with Skype traffic.

VI. Discussion

A. Beyond OS Identification

The algorithms proposed in this paper are not limited to identification of OSes only. Instead, they can be used to identify other aspects of traffic, for example the application that generates the traffic. The algorithm itself does not need to be fundamentally modified.

A simple example of this is illustrated in Figure 7, where we use the Spectrum Selection Algorithm to identify applications.

We apply the Spectrum Selection Algorithm to identify applications using the same traffic traces described in Section V without distinguishing traffic from different operating systems during training. The applications to be detected are YouTube, Skype, and Downloading. Figure 7 shows the performance of the application identification along with the false-positive and false-negative identification rates for each application included in the experiment. We note that the detection rate can reach 85% with 15 minutes of traffic traces. We infer that the Spectrum Selection Algorithm can successfully extract application-specific frequency components, which can be used for simple cases of application identification.

B. 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 full spectrum algorithm only removes the DC component and uses the rest frequency components as the *characteristic frequency components*. So there is no need for the full spectrum algorithm to pick out *characteristic frequency components* from a spectrum with training required for the spectrum selection algorithm. In other words, the training time for full spectrum algorithm is essentially zero.

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.

References

[1] Ts 33.401: System architecture evolution (sae); security architecture. network, ver.11.2.0, release 11. Technical report, 3rd Generation Partnership Project (3GPP), 2011.

[2] 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.

[3] 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.

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

[5] 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.

[6] M. Conti, L. V. Mancini, R. Spolaor, and N. V. Verde. Analyzing android encrypted network traffic to identify user actions. *IEEE Transactions on Information Forensics and Security*, 11(1):114–125, Jan 2016.

[7] Q. Do, B. Martini, and K.-K. R. Choo. Exfiltrating data from android devices. *Computers & Security*, 48:74 – 91, 2015.

[8] Q. Do, B. Martini, and K. K. R. Choo. A data exfiltration and remote exploitation attack on consumer 3d printers. *IEEE Transactions on Information Forensics and Security*, 11(10):2174–2186, Oct 2016.

[9] C. D'Orazio and K.-K. R. Choo. An adversary model to evaluate DRM protection of video contents on ios devices. *Computers & Security*, 56:94 – 110, 2016.

[10] C. J. D'Orazio and K.-K. R. Choo. A technique to circumvent ssl/tls validations on ios devices. *Future Generation Computer Systems*, 2016.

[11] C. J. D'Orazio, R. Lu, K.-K. R. Choo, and A. V. Vasilakos. A markov adversary model to detect vulnerable ios devices and vulnerabilities in ios apps. *Applied Mathematics and Computation*, 293:523 – 544, 2017.

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

[13] 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.

[14] 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.

[15] 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.

TABLE IV
EMPIRICAL RUNNING TIMES FOR THE FULL SPECTRUM, SUPPRESSION, SPECTRUM SELECTION AND HYBRID OS IDENTIFICATION 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

[16] 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.

[17] 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.

[18] 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.

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

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

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

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

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

[24] 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.

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

[26] E. Rocha, P. Salvador, and A. Nogueira. Classification of hidden users' profiles in wireless communications. In K. Pentikousis, R. Aguiar, S. Sargento, and R. Agüero, editors, *Mobile Networks and Management*, pages 3–16, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[27] N. Ruffing, Y. Zhu, R. Libertini, Y. Guan, and R. Bettati. Smartphone reconnaissance: Operating system identification. In *Proceedings of the 13th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, pages 21–21, Berkeley, CA, USA, 2016. USENIX Association.

[28] 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.

[29] T. Stöber, M. Frank, J. Schmitt, and I. Martinovic. Who do you sync you are?: Smartphone fingerprinting via application behaviour. In *Proceedings of the Sixth ACM Conference on Security and Privacy in Wireless and Mobile Networks*, WiSec '13, pages 7–12, New York, NY, USA, 2013. ACM.

[30] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic. Appscanner: Automatic fingerprinting of smartphone apps from encrypted network traffic. In *2016 IEEE European Symposium on Security and Privacy (EuroS P)*, pages 439–454, March 2016.

[31] 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.

[32] Q. Wang, A. Yahyavi, B. Kemme, and W. He. I know what you did on your smartphone: Inferring app usage over encrypted data traffic. In *2015 IEEE Conference on Communications and Network Security (CNS)*, pages 433–441, Sept 2015.

[33] 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.

[34] 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.

[35] Y. Zhu and R. Bettati. Compromising anonymous communication systems using blind source separation. *ACM Trans. Inf. Syst. Secur.*, 13(1):8:1–8:31, Nov. 2009.

[36] Y. Zhu, X. Fu, B. Gramham, R. Bettati, and W. Zhao. Correlation-based traffic analysis attacks on anonymity networks. *IEEE Transactions on Parallel and Distributed Systems*, 21(7):954–967, 2010.

Ye Zhu received the BSc degree in 1994 from Shanghai JiaoTong University and the MSc degree in 2002 from Texas A&M University. He received the PhD degree from the Electrical and Computer Engineering Department at Texas A&M University. He is currently an associate professor in the Department of Electrical and Computer Engineering at Cleveland State University. His research interests include network security, traffic engineering, and wireless sensor networks. He is a member of the IEEE and the IEEE Computer Society.

Nicholas Ruffing received the BS in Computer Engineering and the MS in Electrical Engineering from Cleveland State University - Cleveland, Ohio. He is a currently a software developer for Hyland, creator of OnBase®. His research interests are in network and mobile device privacy and security.

Jonathan Gurary received the BS in Computer Engineering and the MS in Electrical Engineering from Cleveland State University - Cleveland, Ohio. He is a currently a PhD candidate at Cleveland State University. His research interests are in human-computer interaction, especially interfaces in security applications, and network security and privacy for mobile devices.

Yong Guan is an Associate Professor of Electrical and Computer Engineering, and Associate Director for Research, Information Assurance Center at Iowa State University. He is also an Ames Lab associate for Midwest Forensics Resource Center at U.S. DoE's Ames Lab. He received his Ph.D. degree in Computer Engineering from Texas A&M University in 2002, MS and BS degrees in Computer Science from Peking University in 1996 and 1990, respectively. With the support of NSF, IARPA, ARO, and Boeing, his research focuses on security, privacy, and digital forensics. He served as the general chair of 2008 IEEE Symposium on Security and Privacy, co-organizer for ARO Workshop on Digital Forensics, General/Program Chairs for digital forensics workshops/conferences, and coordinator of Digital Forensics WG at NSA/DHS CAE IA Principals Meetings, and editor of IEEE Transactions on Information Forensics and Security and IEEE Transactions on Wireless Communication (Security area). Dr. Guan has been recognized by ISU Award for Early Achievement in Research, Litton Industries Professorship, NSF Career Award, and Outstanding Community Service Award of IEEE Technical Committee on Security and Privacy. He is a senior member of IEEE.

Riccardo Bettati is Professor in the Department of Computer Science at Texas A&M University, where he has been leading the Real-Time Systems Research Group and until 2015 the Center for Information Assurance and Security. He received his Diploma in Informatics from the Swiss Federal Institute of Technology (ETH), Zuerich, Switzerland, in 1988 and his Ph.D. from the University of Illinois at Urbana-Champaign in 1994. From 1993 to 1995, he held a postdoctoral position at the International Computer Science Institute in Berkeley and at the University of California at Berkeley.

His research interests are in traffic analysis and privacy, real-time distributed systems, real-time communication, and network support for resilient distributed applications.

He was the Program and General Chairs of The IEEE Real-Time and Embedded Technology and Applications Symposia in 2002 and 2003, respectively. He shares Best Paper awards with collaborators and students in the IEEE National Aerospace and Electronics Conference and in the Euromicro Conference on Real-Time Systems.