A FRAMEWORK FOR SYSTEM FINGERPRINTING

A Thesis
Presented to
The Academic Faculty

by

Sakthi Vignesh Radhakrishnan

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in the
School of Electrical and Computer Engineering

Georgia Institute of Technology
May 2013

A FRAMEWORK FOR SYSTEM FINGERPRINTING

To my parents,

Radhakrishnan and Vijayakumari,

my family, and all my friends,

without whom none of my success would be possible.

ACKNOWLEDGEMENTS

This thesis would not have been possible without the help and guidance of several individuals who in one way or the other contributed to completion of this study.

First and foremost, I would like to express my sincere gratitude to my advisor Prof. Raheem A Beyah for his continuous support, patience, motivation, and enthusiasm throughout my thesis. I feel deeply honored to have worked with someone who's care and support extends beyond the boundaries of academics.

My sincere thanks also goes to Dr. Selcuk Uluagac who has been a great company throughout my life at Georgia Tech. Without him, I would have missed out on so many opportunities that I now relish. I can never thank him enough for his help and support all through my thesis, especially before deadlines, and on sleepless nights.

Besides, I would like to thank my committee members, Prof. John Copeland and Prof. Henry Owen for their helpful comments and support.

I also extend thanks to my lab buddies, Supreeth, Ram and Venkat for being such an awesome company through all the ups and downs. Without the continuous music from harman/kardon (must mention that its a loop of 10), ever improving sessions of table tennis, tons of humor, breaking open computers, and the QR code, my life at Tech wouldn't have been anywhere close to what I have just experienced. My gratitude also extends to Sang Shin, Troy, Aaron, Marco and Xiaojing for their continuous support and technical guidance. I would also like to extend a special thanks to my friend Gopi Sundaresan who has offered enormous help during all phases of my master's.

Last but not the least, I would like to thank my entire family back in India, my teacher Bhanu, and Subbarayan & family for their continuous support and encouragement. Not to forget, a big hearty thanks to Anand, Priyadarshini, Raj and Priya for their immense love and care during my days away from home.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

The primary objective of the proposed research is to develop a framework for smart and robust fingerprinting of networked systems. Many fingerprinting techniques have been proposed in the past, however most of these techniques are designed for a specific purpose, such as Operating System (OS) fingerprinting, Access Point (AP) fingerprinting, etc. Such standalone techniques often have limitations which render them dysfunctional in certain scenarios or against certain counter measures. In order to overcome such limitations, we propose a fingerprinting framework that can combine multiple fingerprinting techniques in a smart manner, using a centralized decision making engine. We believe that any given scenario or a counter measure is less likely to circumvent a group of diverse fingerprinting techniques, which serves as the primary motivation behind the aforementioned method of attack. Another major portion of the thesis concentrates on the design and development of a device and device type fingerprinting sub-module (GTID) that has been integrated into the proposed framework. This sub-module is designed using the core technique proposed by the authors of [1]. The technique proposed in [1] involves the use of statistical analysis of packet inter arrival times (IATs) to identify the *type* of device that is generating the traffic. However, the work presented in [1] only provides a preliminary study of feasibility and is limited to fingerprinting device types, and not devices. In this work, we analyze the performance of the identification technique on a real campus network and propose modifications that use pattern recognition neural networks to improve the overall performance. Additionally, we impart new capabilities to the fingerprinting technique to enable the identification of *'Unknown'* devices (i.e., devices for which no signature is stored), and also show that it can be extended to perform both device and device type identification.

CHAPTER I

INTRODUCTION

Cyber infrastructure has enabled major revolutions all over the world. We can now communicate, collaborate, and conduct business transactions with anyone in the world at any time. Cyber space has quickly gone from being reserved for scientists (i.e., ARPANET) to something that is critical to enabling basic functions in everyday life (e.g., vehicle navigation, banking, commerce). Cyber space is not just expanding, but in fact is getting more dense with the introduction of smart phones and tablets. With BYODs (Bring Your Own Devices) becoming the norms at campuses and corporates, the number of active devices on local area networks and the Internet has grown exponentially. Given the increasingly critical nature of cyberspace, it is imperative that it is secured.

One of the most significant threats to today's computer networks is the threat from insiders [2]. Insider attacks are dangerous because they subvert the traditional defense mechanisms and are initiated "behind" them. Further, they are initiated by individuals who have valid credentials to access the network and systems. In a recent National Security Survey [3] it was pointed out that forty percent of all incidents reported by the 7,818 respondents (representing 36,000 US businesses) were attributed to insiders. Further, nearly seventy-five percent of all cyber theft was attributed to insiders. Insider attacks can take on many meanings, but they can be broadly classified as: 1) *Malfeasors* - Insiders that do not have malicious intent and can unknowingly bring harm to an institution (e.g., a young private connecting a video game system to a military network) or 2) *False Insider* - An individual who is strategically planted inside of an institution (or planted devices) in order to cause harm to the institution.

Given that traditional perimeter defense mechanisms (e.g., firewalls, signature- and anomaly-based NIDS) are not effective against insider attacks, many institutions, have begun to invest in technologies that have the ability to detect insiders. Network Access

Control (NAC) is one of the popular approaches used by many organizations to prevent unauthorized access to their network. In general, such a system consists of a NAC clients that uses port-based authentication (802.1x) in conjunction with a backend authorization server (e.g., RADIUS server). While such a solution can provide reasonable security when clients are installed on the device it is to protect, it has limited to no support for other networked devices (e.g., Internet Protocol (IP) phones [4]). Moreover, with the move towards an Internet of things and with a constantly evolving threat landscape, it is unlikely that vendors will develop and support NAC clients for dozens of architectures and OS variants.

This lack of support for architecture and OS diversity can be seen today, with popular vendors NAC solutions (e.g., McAfee, Symantec, Cisco) only fully supporting Windows and just recently Mac and Linux OSs. A point of attack for a skilled attacker is to insert a device that appears to be a printer (or any other unmanageable device) on the network. This threat is especially heightened when we consider that insiders have the correct credentials to access the network. Thus, in addition to defending against malicious processes or components of known trusted devices with NAC clients, a solution is needed to ensure that rogue devices cannot be inserted into the network (posing as unmanageable nodes), even if the insertion is executed by an insider who possesses the proper network credentials.

1.1 Research Objective

The objective of this thesis is to 1) develop a modular fingerprinting framework that can be used to integrate a wide variety of fingerprinting techniques such as OS fingerprinting, device fingerprinting, etc.; 2) analyze in detail and improve an existing device fingerprinting technique; 3) implement this technique as a deployable module (GTID) and integrate it into the framework, along with existing techniques such as OS fingerprinting, etc.

The general trend with the works published in the area of fingerprinting is that, they are all stand alone techniques designed specifically for a single type of fingerprinting (e.g., OS fingerprinting). Such standalone techniques have some limitations which render them less useful in one or more scenarios. This research is novel in that it presents a fingerprinting framework that can be used to combine a variety of such standalone fingerprinting techniques to improve the overall robustness and performance of system fingerprinting. In addition to this, the thesis proposes a device and device type identification technique (GTID) with significant analysis and improvements to address the limitations of an existing technique described in [1].

1.2 A Summary of Contributions

This thesis is an encapsulation of two major contributions. The first contribution is the development of a deployable framework for system fingerprinting and the second contribution is GTID, a technique for device and device type fingerprinting. These contributions are explained below.

1.2.1 Fingerprinting Framework

The framework's design focuses on ways to combine multiple fingerprinting techniques such that the overall strength is greater than the cumulative strengths of the standalone techniques. The proposed framework has three major modules. The first module is called the *Monitoring-Module*, which consists of tools that aid in the process of setting up network interfaces cards and capturing network packets. The framework currently supports Wifi and Ethernet interfaces, however the framework is modular and can be extended to support other interfaces such as Universal Serial Bus (USB), Universal Software Radio Peripheral (USRP) and more. The *Identification-Module* houses a list of identification sub-modules that can work independently and feed results to the next stage. These sub-modules use the monitoring tools provided in the Monitoring-Module to sniff packets or to send out specially crafted traffic to the target system. At present, three sub-modules have been implemented in the Identification-Module. The first is OS fingerprinting, which uses Nmap [5] as the underlying program. The second sub-module is *GTID*, which has been implemented from scratch. This sub-module will serve the purpose of fingerprinting devices and device types. MAC-lookup is another simple sub-module that has been implemented. This uses IEEE's OUI List [6] in addition to a XML database to map MAC address to hosts/make. The third and final module is the *Decision-Module* which will collect and processes the results fed in by the Identification-Module. Using the stored results, the decision module will be able to send useful feedback to the identification tools. For example, OS results from the Nmap sub-module can be sent as feedback to GTID, which can then wisely select a subset of the available signatures. This will in turn help in reducing the number of false positives. This framework also provides a graphical user interface that can be used to select target hosts, apply identification techniques and to present results.

1.2.2 GTID: A Technique for Physical Device and Device Type Fingerprinting

GTID is a technique for device and device type fingerprinting. This work was motivated by [1], which uses information leaked by the physical implementation of a device through its network traffic to identify a device and a device's type. This technique relies on the use of statistical analysis to measure time-variant behavior in traffic and to create unique, reproducible device and device type signatures. However, the work presented in [1] has the following limitations that we address through this contribution.

Limitations:

1) The work presented in [1] only provides a preliminary study of feasibility and does not perform extensive analysis on real networks. Such analysis is important given the fact that inter arrival times (IATs) can be easily affected by both, physical and MAC layer contention.
2) Their proposed technique provided average results when used to analyze traffic on a real network. This has been improved using a new technique that involves the use of pattern recognition neural networks.
3) The original technique does not have the capability to identify unknown devices/types (i.e., devices/types for which no signature exists in the master database). This has been made possible through this work.
4) The authors of [1] focus only on wireless networks, however this work extends the analysis to wired networks.

In addition to the above contributions, this work also looks into the effect of clock skews on network traffic, possible counter measures to the proposed technique and into a deployable implementation that can be integrated into the Identification-Module of the proposed framework. The primary weakness of this technique, as with most works that rely on fine-grained packet timing, is that the timing is lost as a result of buffering in switches and routers. Therefore, this technique is not suited for identification across the Internet. Rather, it is perfectly suitable for the significant challenge of local network access control.

1.3 Thesis Outline

The remainder of this thesis proceeds as follows. In, Chapter 2 the related works are presented followed by a description of the framework in Chapter 3. GTID is discussed in Chapter 4, and the thesis concludes in Chapter 5 with discussions on the future work.

CHAPTER II

RELATED WORK

The existing work in this area can be placed into four broad categories, namely, OS fingerprinting, device type fingerprinting, physical device fingerprinting and application fingerprinting.

The work in the category of OS fingerprinting, differentiates different OSs based on different features of the protocol stack. Nmap [5] and Xprobe [7, 8] are active fingerprinting tools that create special test packets to determine which OS is being used by the target. In contrast to the aforementioned active approaches, p0f [9] uses TCP/IP protocol information to passively determine the OS. SinFP [10] is an active and passive OS fingerprinting technique that bypasses some of the emerging limitations of Nmap (i.e., working with PAT/NAT configurations and emerging packet normalization technologies). Such techniques can be used by an adversary to launch OS specific attacks on target systems or can be used for automating security tests in industries that use Supervisory Control and Data Acquisition (SCADA) systems [11]. However OS fingerprinting techniques do have their own limitation. The authors of [12] claim that there are four major factors that significantly limit the usefulness of automated OS fingerprinting techniques. First, is over fitting of fingerprints to non-OS-specific behavioral differences; second, is in-distinguishability of different OS variants; the third, is biasing of an automatic tool to the makeup of the training data; and the fourth, is lack of ability for an automatic tool to exploit protocol and software semantics. In addition to having natural limitations, OS fingerprinting techniques can also be evaded using counter measures like the ones discussed in [13, 14]. Although the methods described above are quite effective for OS fingerprinting, the goal of the GTID sub-module is to differentiate device hardware and device types. Also, since our framework combines GTID and OS fingerprinting, it has the combined advantage of both techniques which renders it useful for both OS and device hardware fingerprinting.

The second category of fingerprinting is host or physical device fingerprinting. Seminal work in this area was introduced by Kohno et al. in [15]. In [15], a method for remotely fingerprinting a physical device by exploiting the implementation of the TCP protocol stack was proposed. The authors use the TCP timestamp option of outgoing TCP packets to reveal information about the sender's internal clock. The authors' technique exploits microscopic deviations in the clock skews to derive a clock cycle pattern as the identity for a device. The authors of [16] take a similar approach of that in [15] (i.e., using clock skew to uniquely identify nodes), however the goal of [16] is to uniquely fingerprint APs. Also, instead of getting the timestamp from TCP packets, they obtain the timestamp from 802.11 beacon frames. Because of the use of the 802.11 beacons, this approach only works for AP fingerprinting, whereas GTID is capable of working for general devices such as laptops, tablets and cell phones. There have also been physical layer approaches to fingerprint wireless devices. Radio frequency (RF) emitter fingerprinting uses the distinct electromagnetic (EM) characteristics that arise from differences in circuit topology and manufacturing tolerances. This approach has a history of use in cellular systems and has more recently been applied to Wi-Fi [17] and Bluetooth [18] emitters. The EM properties fingerprint the unique transmitter of a signal and differs from emitter to emitter. Such techniques require expensive signal analyzer hardware to be within RF range of the target. In contrast, GTID, and thus our framework, requires only a network tap at a switch to capture traffic on a wired segment that could be a hop downstream. Moreover, GTID fingerprints devices independent of the protocol and does not require deep packet inspection (e.g., timestamps). In contrast to [15], the GTID sub-module is better positioned for scalability, does not compromise privacy and works on IP level encrypted streams. In general, a major difference between the works in this category and GTID is that the techniques presented here only fingerprint devices, not both devices and device types. Also, the proposed technique will be able to classify a previously unseen device to be *Unknown*, which is an unique feature not present it other techniques discussed above. Given that GTID is a sub-module of the proposed framework, it too shares the above advantages in addition to being able to fingerprint OSs, etc.

Another body of work that is very important is device type fingerprinting. The main objective of the techniques in this category is to be able to remotely identify a specific device type. In [19], the authors fingerprint wireless AP types by probing them with various regular and malformed packets. Along similar lines, the work discussed in [20], introduces a device fingerprinting technique that can detect the type of wireless AP that a traffic stream passes through. It relies on distinct patterns (from the hardware composition of the device) that are generated in the network traffic as a result of specially crafted data streams. Wavelets were used to extract the signatures of the device types from the data stream. Both of these are active techniques and have the limitation that they can only fingerprinting wireless AP types. The authors in [21] use timing information between commands and responses on the Universal Serial Bus (USB) to distinguish between variations in model identifiers, OSs (and sometimes OS version number), and whether a machine is answering from a real or virtual environment. The obvious limitation of this work is that it requires one to be in physical possession of the device. The authors of [22] propose a technique for device type behavioral and temporal fingerprinting. They model a specific protocol implementation (i.e., the Session Initiation Protocol - SIP) and create a behavioral fingerprint using a Temporal Random Parameterized Tree Extended Finite State Machine (TRFSM). Their technique can learn distinctive timing patterns of the transitions in the SIP protocol's state machine. These timing patterns for the state machine can be detected by observing the difference between various outgoing and incoming SIP messages of the device being fingerprinted. In their early work [23], their technique required knowledge of the entire syntax of the protocol. However, in [22] this requirement is relaxed as they only need a corpus containing SIP sessions. The authors of [22, 23] develop a real-time approach and discuss deploying their techniques in [24]. The disadvantage of this technique is that it is limited to a specific application layer protocol - SIP (similar to the limitations of [15, 16]). The technique proposed in [1] uses the differences in the statistical characteristics of inter arrival times of the packets to identify the type of device that is generating the traffic. Such an approach was shown [1] to overcome some of the limitations faced by other techniques in this category of fingerprinting. Moreover, this technique has also been shown to work across different traffic types, which means it is not protocol dependent akin to the techniques proposed in

[15, 16, 25]. Finally, due to the nature of the core feature that is used, this technique is able to work across a variety of device types, and is not limited to a specific type of device, which is the case in [19, 20]. In general, a major difference between the work in this category and the GTID sub-module is that the techniques presented here only fingerprint device types, not both devices and device types. Moreover, GTID also address some of the limitations of the work presented in [1]. A detailed explanation of the drawbacks and our method of attack is explained in Chapter 4. Since, GTID is a part of the proposed framework, the framework will be able to fingerprint both physical devices and device types in addition to fingerprinting OSs, etc.

The next class of fingerprinting deals with the identification of application traffic. The techniques proposed here can be further sub-classified as session-based, content-based and constraint-based techniques. The session-based and content-based methods require preparation and training before the technique can be made ready to use. In addition to that, the signature details need to be updated as and when the protocol or application behavior changes. Well known port matching can be seen as a simple example of a session-based traffic identification technique. Here the traffic of interest will be mapped to an application based on the details provided in [26]. More advanced techniques in this area are discussed in [27, 28]. The content-based techniques try to identify payload contents that are static and unique for a given application. This is then used as a signature to classify unknown traffic. The problem with such an approach is that it may fail with payloads that are encrypted or when the application undergoes a version update. The constraint-based traffic identification method can be seen as a subclass of session-based techniques. The major difference of the constraint-based methods is that they do not require any application-level protocol information. These methods use statistical techniques such as statistical signatures [29] or supervised machine learning [30] to map traffic into a specific set of predetermined clusters. These clusters are usually formed by applying constraints on traffic characteristics such as flow duration, average packet size of a flow, packet inter arrival time and more. Clearly, this category of work is orthogonal to the sub-module GTID which focuses on fingerprinting physical devices and device types. However, these techniques can still be very useful for fingerprinting networked systems and might be integrated as a part of our framework in the future.

Almost all the fingerprinting techniques that we have discussed above are standalone techniques and this has been the general trend with the works published in this area of research. The framework for system fingerprinting in general and the design ideas on integrating multiple standalone fingerprinting techniques, are novel, and to the best of our knowledge there is no prior work that has addressed these specifically. Further, our device and device type identification sub-module (GTID) overcomes several aforementioned limitations of existing device / device type fingerprinting techniques.

CHAPTER III

A FRAMEWORK FOR SYSTEM FINGERPRINTING

The proposed framework is discussed in this chapter with insights into the motivation behind this contribution and scenarios where such a framework can be useful.

3.1 Motivation

As discussed in Section 2, most of the works published in this area of research have concentrated in the development of standalone techniques. The design goals of such techniques are focused primarily on a single type of fingerprinting (e.g., device fingerprinting). While such an approach is good for serving the needs of a particular application, it cannot be used for broader needs such as fingerprinting networked systems. For example, an OS fingerprinting technique provides very little or no information related to a system's hardware. Also, each technique has one or more limitations that renders them less useful in certain scenarios. For example, the RF-based device fingerprinting technique requires the target system to be in wireless range of the detection device. This is not always possible, and in such cases one could use alternate techniques proposed in [15] to fingerprint devices. In addition to this, the targeted system can also use some form of counter measures to evade a fingerprinting technique. As an example, a simple MAC lookup technique can be easily evaded by an attacker who spoofs the MAC address of his network interface card. However, it is very unlikely that a given scenario or a counter measure will evade a group of fingerprinting techniques, each of which uses a different detection algorithm. This serves as the primary motivation for developing a fingerprinting framework that can integrate multiple standalone techniques in a manner, where the overall strength is greater than the cumulative strengths of individual techniques.

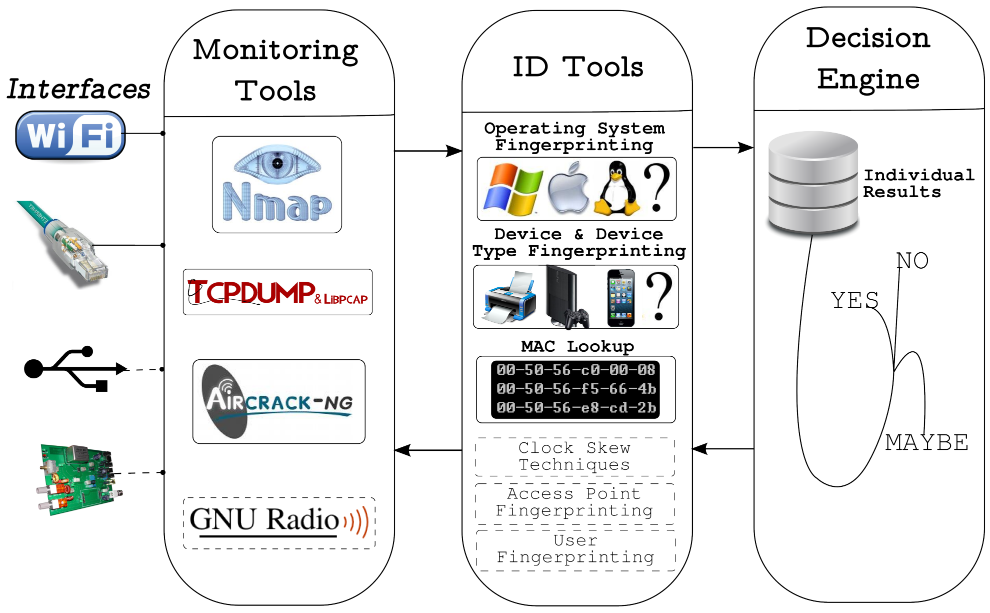


Figure 1: Overview of the Framework

3.2 Overview of the Framework

As mentioned earlier, the central objective of this framework is to enable the integration of various fingerprinting techniques that are available now and also those that will be available in the future. To achieve this, the framework has been modularized into three segments: the Monitoring-Module, the Identification-Module and the Decision-Module. An overview of the framework is shown in Figure 1. The solid boxes in Figure 1 refer to modules that have been currently integrated, and those in dotted boxes show possible extensions. The following sections discuss in detail each of these modules and the components that have been currently integrated into the framework.

3.3 Monitoring Module

The Monitoring-Module performs two primary tasks. The first task is to monitor network traffic in one of the supported interfaces and the second is to generate and send out specially crafted packets. This may include packets with abnormal flags, SYN packets, etc. *Tcpdump* [31] is currently integrated into this module to enable the monitoring of traffic from within the framework. Tcpdump is a popular packet analyzer that can be used to intercept and capture packets on interfaces such as that for Ethernet. The next tool that is integrated into this module is *Nmap* [5]. Nmap is a popular open source OS fingerprinting tool used for network exploration and hacking. It is capable of both probing as well as fingerprinting, however the probing capabilities of Nmap are most relevant to this module.

In addition to housing techniques for monitoring and probing, this module also brings with it tools that are required to set up the network interfaces. As shown in Figure 1, this is the only module that is in direct contact with the network interfaces. One such tool that is currently integrated into this module is *Airmon-ng* [32]. This tool aids in setting up virtual interfaces and in configuring wireless NICs to monitor traffic in promiscuous mode. This brings us to the list of interfaces that are currently supported by this framework. As shown in Figure 1, the framework currently supports Ethernet and Wifi interfaces. However this can be extended in the future to include Universal Serial Bus (USB), Software Defined Radios, bluetooth, etc. Such an expansion will also aid identification techniques such as [21] and [17] respectively.

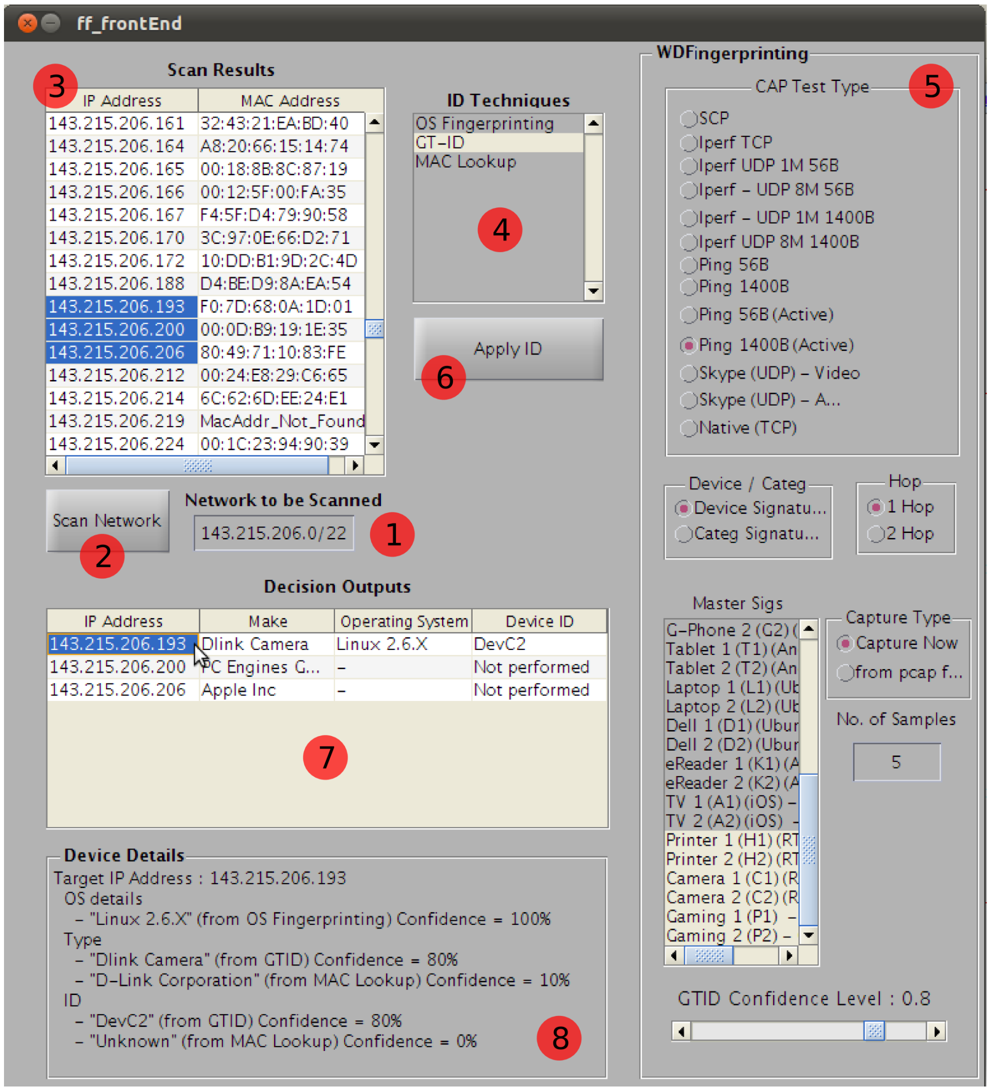
3.4 Identification Module

The Identification-Module is the heart of this framework. This module is responsible for performing different types of fingerprinting and houses a number of identification techniques. Currently, three such techniques have been integrated into this module. The first is GTID, a tool that was developed as a part of this thesis. This tool is capable of both device and device type identification. It uses trained neural networks to recognize previously seen timing patterns in network flows, to identify both the device and the device's type. A more detailed explanation and analysis of this tool can be found in Chapter 4. The next tool is a simple MAC look up module, which was also developed as a part of this work. It uses IEEE's OUI list [6] to map a particular MAC address to a company or make. This sub-module also has a simple XML database which can be used to store known MAC addresses and information related to it. The tool queries both places, with higher priority on the custom built XML database. The third and final sub-module that is currently supported is OS Fingerprinting. This sub-module uses Nmap [5] as the underlying program to fingerprint OSs. It has its own database of signatures that it uses to match the OS of the target host. As observed from the above discussion, each fingerprinting sub-module is completely independent from one another, and uses different techniques and separate databases. One of the main objectives of the framework is to derive a way to integrate such stand-alone modules in a way that they can work hand-in-hand. The way this is done is by using a centralized Decision-Engine which is briefed in the next section.

3.5 Decision Module

This module aids in combining techniques that are available in the Identification-Module in a way that helps them work together. It houses a centralized database that holds the results obtained from each of the ID techniques. These results are properly categorized and mapped to their respective technique. For example, MAC look up and GTID give out the results for device ID. In this case both of these results are categorized under device ID and each of them will be mapped to the respective technique. This helps in implementing weights which represent how dependable the results are from a particular technique. The GUI provides users control over the weights which are customizable. As an alternative, one may chose a network scenario, from a list, which auto configures the weights according to the selected scenario. For example, if the user is very sure about the integrity of the network, he or she can set a high weight for MAC lookup (using the customize option) or use a network scenario from the provided list that closely reflects the network's climate.

As shown in Figure 1, the decision engine has a feed back path that is used for conveying information back to the Identification-Module. Using this, the decision engine will be able to send back information that aids in the trimming of master databases used by the identification sub-modules. For example, when one applies OS fingerprinting on a specific target, the decision engine caches this information. Now, when GTID is invoked, the decision engine passes the OS information to GTID, which will then be able to cherry pick the list of masters signatures that run that specific OS. This has two major advantages. First, it helps in improving the accuracy of the identification techniques (the smaller the number of master signatures, the smaller the probability of false positives). Second, it helps in reducing the identification times of the techniques (the smaller the number of masters signatures, the smaller the processing time). Currently, this feature has been incorporated into GTID and



Scan Results

IP Address	MAC Address
143.215.206.161	32:43:21:EA:BD:40
143.215.206.164	A8:20:66:15:14:74
143.215.206.165	00:18:8B:8C:87:19
143.215.206.166	00:12:5F:00:FA:35
143.215.206.167	F4:5F:D4:79:90:58
143.215.206.170	3C:97:0E:66:D2:71
143.215.206.172	10:DD:B1:9D:2C:4D
143.215.206.188	D4:BE:D9:8A:EA:54
143.215.206.193	F0:7D:68:0A:1D:01
143.215.206.200	00:0D:B9:19:1E:35
143.215.206.206	80:49:71:10:83:FE
143.215.206.212	00:24:E8:29:C6:65
143.215.206.214	6C:62:6D:EE:24:E1
143.215.206.219	MacAddr_Not_Found
143.215.206.224	00:1C:23:94:90:39

ID Techniques
- OS Fingerprinting
- GT-ID
- MAC Lookup

Network to be Scanned: 143.215.206.0/22

Decision Outputs

IP Address	Make	Operating System	Device ID
143.215.206.193	Dlink Camera	Linux 2.6.X	DevC2
143.215.206.200	PC Engines G...	-	Not performed
143.215.206.206	Apple Inc	-	Not performed

Device Details

Target IP Address : 143.215.206.193
OS details
- "Linux 2.6.X" (from OS Fingerprinting) Confidence = 100%
Type
- "Dlink Camera" (from GTID) Confidence = 80%
- "D-Link Corporation" (from MAC Lookup) Confidence = 10%
ID
- "DevC2" (from GTID) Confidence = 80%
- "Unknown" (from MAC Lookup) Confidence = 0%

WDFingerprinting

CAP Test Type: SCP / Iperf TCP / Iperf UDP 1M 56B / Iperf - UDP 8M 56B / Iperf - UDP 1M 1400B / Iperf UDP 8M 1400B / Ping 56B / Ping 1400B / Ping 56B (Active) / [✓ Ping 1400B (Active)] / Skype (UDP) - Video / Skype (UDP) - A... / Native (TCP)

Device / Categ: [✓ Device Signatu...] / Categ Signatu...

Hop: [✓ 1 Hop] / 2 Hop

Master Sigs
- G-Phone 2 (G2) (
- Tablet 1 (T1) (An
- Tablet 2 (T2) (An
- Laptop 1 (L1) (Ub
- Laptop 2 (L2) (Ub
- Dell 1 (D1) (Ubur
- Dell 2 (D2) (Ubur
- eReader 1 (K1) (A
- eReader 2 (K2) (A
- TV 1 (A1) (iOS) -
- TV 2 (A2) (iOS) -
- Printer 1 (H1) (RT
- Printer 2 (H2) (RT
- Camera 1 (C1) (R
- Camera 2 (C2) (R
- Gaming 1 (P1) -
- Gaming 2 (P2) -

Capture Type: [✓ Capture Now] / from pcap f...

No. of Samples: 5

GTID Confidence Level : 0.8

Figure 2: Screen-shot of framework prototype

will be extended to other techniques. The decision engine also interfaces with the GUI to display the final results in a very concise manner. It uses the provided weight and confidence measure of the result to compute the overall confidence of the output generated by a particular identification technique. For example, if the user assigns a weight of 60% to GTID and if GTID produces a result with 80% confidence, then the overall confidence of that particular outcome or result is measured as the product of the two confidence values, which is 48%. Finally, the Decision-Module uses these confidence measures to display the results for each system parameter (such as OS, ID, Type, etc.).

Figure 2 shows a screen-shot of the framework prototype that was developed. The way this operates is as follows. At first, the user types in the network address that is to be scanned in the input box (marked 1). Then the user clicks “Scan Network” (marked 2) which tabulates the list of devices (marked 3) that are up in the scanned network. The user can now select a target system from the list shown under “Scan Results” (marked 3) and apply any of the available ID techniques (marked 4). When an ID technique is selected, the tool displays related options on the right (marked 5). The screen-shot in Figure 2 shows the options that are available for GTID sub-module, which includes a slide-bar for setting the confidence level. After setting up the ID technique, the user clicks “Apply ID” (marked 6) which gives out the ID result under “Decision Outputs”. The screen-shot in this case shows *OS Fingerprinting* and *MAC Lookup* applied on three targets, and *GTID* applied on the first target alone. The outcomes listed under “Decision Outputs” (marked 7) are concise results generated by the decision algorithm. To view detailed results about a specific target, the user clicks the target under “Decision Outputs” (marked 7) and this generates a more elaborate result under “Device Details” (marked 8).

CHAPTER IV

AN IDENTIFICATION SUBMODULE FOR DEVICE AND DEVICE TYPE FINGERPRINTING

4.1 Motivation

An important requirement in managing a network is to be able to identify the nodes that are connected or attempting to connect. This is required to implement NACs that authenticate devices using software clients, login credentials or rely on innate characteristics for identification. The problem with using such techniques are 1) software clients - it cannot be installed on all devices (e.g. embedded devices such as IP-Cameras or Sensor Nodes); 2) login credentials - authenticates only the users and not devices, which is not secure (Figure 3); 3) innate characteristics - these parameters can be easily faked (e.g. spoofing of IP Address or MAC Address). To overcome some of these limitations, the authors of [1] proposed a technique to fingerprint device types using statistical analysis on the inter packet arrival (IAT) times. Such an approach is passive (does not require a client software) and is difficult to fake. However in order for the technique to be used for access-control it needs to be able to identify devices and not just the types. Moreover, the work presented in [1] only provides a preliminary study of feasibility and does not perform extensive analysis on real networks. Such analysis are important given the fact that IATs can be easily affected by both, physical and MAC layer contention.

The work presented in this chapter address the above issues through extensive analysis of the technique on a real campus network. During the initial phase of this analysis, it was found that the technique proposed in [1] did not perform well under real network conditions. Thus, a number of other processing techniques in the area of signal processing, machine learning and pattern recognition were evaluated. This analysis revealed the suitability of pattern recognition neural networks, which shows acceptable levels of performance for both device and device type identification. The experimental results obtained

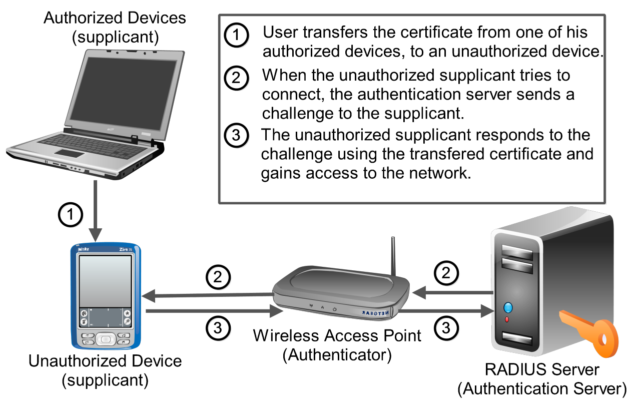

Authorized Devices
(supplicant)
1 User transfers the certificate from one of his authorized devices, to an unauthorized device.
2 When the unauthorized supplicant tries to connect, the authentication server sends a challenge to the supplicant.
3 The unauthorized supplicant responds to the challenge using the transfered certificate and gains access to the network.
1
2
3
2
3
Unauthorized Device
(supplicant)
Wireless Access Point
(Authenticator)
RADIUS Server
(Authentication Server)

Figure 3: Problem with 802.1x

using neural networks show a significant improvement over the original technique. Moreover, for a detection technique to be of practical use, it needs to be quick and accurate. High accuracy with high latency may lead to correct, but less useful results. In order to study the detection speed of this technique, a prototype version of GTID was developed in MATLAB and its performance was tested.

4.2 Background

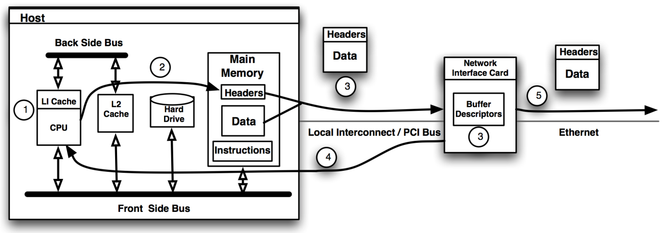


Figure 4: Packet flow in hardware

Device packet creation is a complex process. This process involves many internal parts of the node working together. Before a packet can be sent, the instruction set that initiates the process must be extracted from the memory hierarchy (LI/L2 cache, main memory, hard disk) and sent to the CPU for execution. The OS then directs the CPU to create a buffer descriptor in main memory, which contains the starting memory address and length of the packet that is to be sent. Multiple buffer descriptors are created if the packet consists of multiple dis-contiguous regions of memory. The OS then directs the CPU to write information about the new buffer descriptors to a memory-mapped register on the network interface card (NIC). These data traverse the front side bus through the Northbridge to the PCI bus. The NIC initiates one or more direct memory access (DMA) transfer(s) to retrieve the descriptors. Then, the NIC initiates one or more DMA transfer(s) to move the actual packet data from the main memory into NIC's transmit buffer using the address and length information in the buffer descriptors. These data again leave the front side bus, and travel to the NIC through the Northbridge and the PCI bus. Finally, the NIC informs the OS and CPU that the descriptor has been processed. Then, the NIC sends the packet out onto the network through its medium access control (MAC) unit [33]. A diagramatic representation of this is shown in Figure 4.

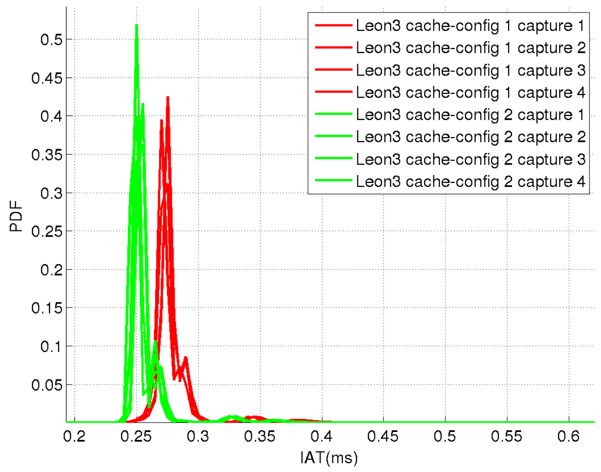


Figure 5: PDFs of IATs for different CPU configurations

Assuming that the effect of the OS can be abstracted, one can see that the major components that affect the creation of packets are: the CPU, L1/L2 cache, physical memory, the direct memory access (DMA) controller, the front side bus, the back side bus, the PCI bus, and the NIC. The opportunities for diversity are both at the device level and at the component level. At the device level, different vendors use different components with different capabilities and algorithms (e.g., Dell Latitude 2110 with Intel Atom N470 processor @ 1.83GHz vs. Lenovo G570 with Intel Core i5-2430 processor @ 2.4GHz) to create a device's internal architecture. Accordingly, the packet creation process varies across architectures aiding device type identification. The author of [34] has studied the variation in IAT patterns for different CPU configurations and clock frequencies, the results of which are presented in Figure 5 and 6. Figure 5 shows the difference in the IAT patterns for two cache configurations (cache-config 1: data cache size 4KB, instruction cache uses LRU replacement algorithm; cache-config 2: data cache size 8KB, instruction cache uses Random replacement algorithm), and Figure 6 shows the result for two different clock frequencies. These experiments clearly demonstrate the impact of hardware configuration on packet IATs.

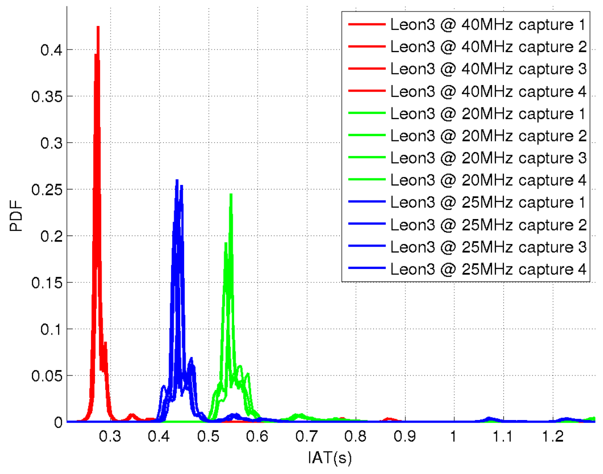


Figure 6: PDFs of IATs for different clock frequencies

Moreover, microscopic differences in the clock frequencies (clock skews) also contribute to the differences in the timing pattern. For example, assume Asus netbooks take x CPU cycles to sleep for 10ms and generate a 56 byte ping packet, and f_1 and f_2 are the clock frequencies of two Asus netbooks, then the difference in the time taken to generate a packet is $x(f_2 - f_1)/(f_1 f_2)$. This value will be reflected as average differences in the inter arrival times (IATs) between two devices. Figure 7 shows a plot of average IAT values, of 5 identical Asus netbooks generating 56 byte ping packets at 10ms intevals. The x-axis denotes the sample numbers, where each sample consists of 2.5K ping packets. For instance, 20 samples will have 50K ping packets in total. The average IAT of packets in each sample is shown on the y-axis. A clear difference in the average IATs of each of the 5 Asus netbooks can be noted, even though they all belong to the same type. These minor variations in the average IATs can affect the frequency count, which in-turn will result in different signatures for each device. These two are among several possible reasons which enables device identification.

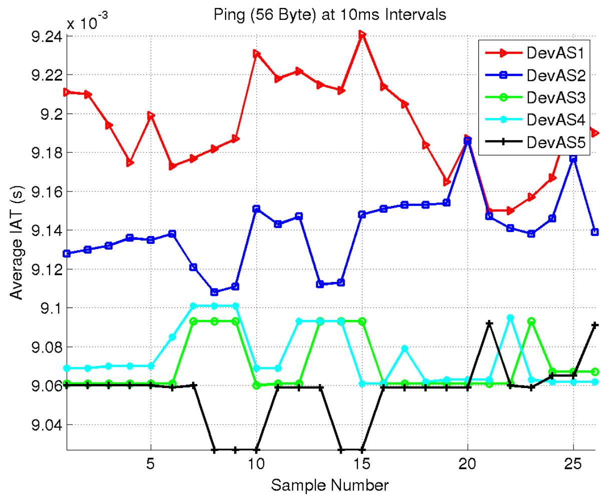


Figure 7: Effect of clock skew on inter arrival times (IATs)

4.3 Overview of the Technique

This section introduces the major components of GTID and discuss the algorithm used to identify devices and their types. Further, it articulates the technical methods that were used in evaluating the overall performance of the technique.

4.3.1 Components of the Technique

In this setup, a wireless device transmits data over the air to an AP. The AP forwards data over its wired interface towards the final destination. GTID passively collects traffic on a wired segment between the AP and the final destination to identify the type of wireless client. As depicted in Figure 8, GTID consists of four major components: *feature extraction, signature generation*, *similarity measure* and *enroll.*

Feature Extraction : As traffic is collected, the feature extraction process measures traffic properties successively in time. Example measurements include inter-arrival time

(IAT), round-trip-time (RTT), and packet size. The resulting feature vector is a time series of values passed to the signature generation process for time-series analysis. When selecting a feature to measure, it should preserve the information pertinent to the type of device and capture discriminating properties for successful classification. This analysis uses the packet IAT as the feature for fingerprinting a physical device. IAT measures the delay (Δt) between successive packets and characterizes the traffic rate. The IAT feature vector is defined as:

$$f = (\Delta\ t_1, \Delta\ t_{2,}, \Delta\ t_3, ..., \Delta\ t_i) \tag{1}$$

where $\Delta\ t_i$ is the inter-arrival time between packet i and $i-1$, and the first collected packet is $i = 0$.

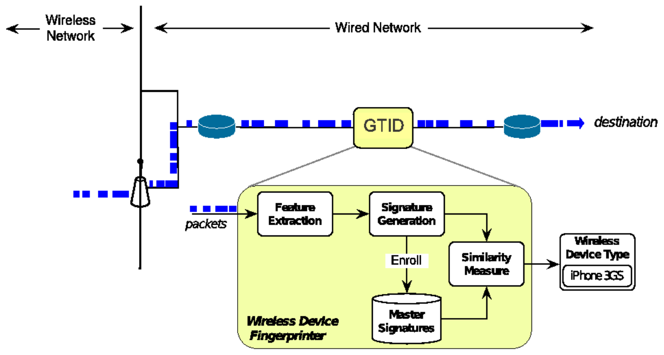


Figure 8: Overview of GTID

Signature Generation : The signature generation process uses statistical analysis to reveal patterns embedded in the traffic measurements. This method adopts a time-domain method for signature generation, which relies on the distribution of the IAT feature vector. Distributions capture the frequency density of events over discrete intervals of time. Due to the periodic nature of network traffic, distributions are a useful tool for traffic analysis. The frequency count is defined as a vector that holds the number of IAT values falling within each of the N equally spaced bins. Dividing the frequency count by the total number of IAT values produces the probability distribution, p, which is the fingerprint. The interval of time for a bin is refered to as its bin width, so the edges of the interval for the nth bin are $(binstart + (n-1) * binwidth)$ and $(binstart + n * binwidth)$, where $binstart$ is the minimum inter arrival time accommodated in the distribution.

Algorithm 1 Device ID and Type Identification

```
1: $Identify - ID - Type()$
2: begin
3:   $\Theta_{ID}, \Theta_{Type}, Dev_{list}, Type_{list} \leftarrow MastersDB(Traffic\ Type)$
4:   $\vec{\lambda_{ID}}, \vec{\lambda_{Type}} \leftarrow MaskingVectors()$
5:   $S \leftarrow IAT\_Sample()$
6:   $\Omega \leftarrow generate\_signature(S)$
7:   $\vec{out}_{ID} \leftarrow \vec{\lambda}_{ID} * \vec{sim}(\Theta_{ID}, \Omega)$
8:   $index, closeness \leftarrow max(\vec{out}_{ID})$
9:   $X \leftarrow 10\_percentile\_TP(Dev_{list}^{index})$
10:  if $(closeness > X)$
11:      return $Dev_{list}^{index}, Corresponding\ Type$
12:  else
13:    $\vec{out}_{Type} \leftarrow \vec{\lambda}_{Type} * \vec{sim}(\Theta_{Type}, \Omega)$
14:    $index, closeness \leftarrow max(\vec{out}_{Type})$
15:    $X \leftarrow 10\_percentile\_TP(Type_{list}^{index})$
16:    if $(closeness > X)$
17:        return $Unknown, Type_{list}^{index}$
18:    else
19:        return $Unknown, Unknown$
20:  end if
21: end
```

The device signature is sensitive to the bin width and different bin widths will reveal different information about the feature vector. Smaller bin widths cause fewer IAT values to occur within a particular bin, and what may appear to be meaningful information may really be due to random variations in the traffic rate. Conversely, larger bin widths may omit important information, aggregating information into fewer bins that might otherwise help to discriminate between two different devices. Based on the experiments, $N = 300$ was emperically determined to be an ideal choice for all traffic types tested in this paper. This value of N is then used to determine the binwidths for each traffic type.

Similarity Measure : Once signatures are generated, it is passed through trained neural networks that are present in the master database. This yields *closeness* values between 0 and 1 for each device in the database (note that 1 denotes a perfect match). These values of *closeness* or *similarity* measures are used to compare an unknown signature to the master signatures, which are essentially a collection of previously seen signatures.

Enroll : Signatures generated in step one are used to train Artificial Neural Networks (ANNs) which registers the pattern and in essence enrolls that device or device type. ANNs are basically computational models inspired from biological neural networks and they imitate them both structurally and functionally. An ANN consists of a group of interconnected computational units called neurons. These neurons take in inputs and transform them according to a specified activation function to generate an output. The technique uses ANNs that belong to a class called *feedforward networks*, where there is only a one-way connection from the input to the output layer. ANNs belonging to this class require supervised training and are commonly used for prediction, *pattern recognition* and nonlinear function fitting. The feedforward ANN is configured to use *scaled conjugate gradient backpropagation* as the training function. The basic backpropagation algorithm adjusts weights in the steepest descent direction (negative of the gradient). This is the direction in which the performance function is decreasing most rapidly. It turns out that, although the function decreases most rapidly along the negative of the gradient, this does not necessarily produce the fastest convergence. In the conjugate gradient algorithms a search is performed along conjugate directions, which generally produces faster convergence than steepest descent directions. It should also be noted that the proposed technique uses sigmoid hidden and output neurons which are ideal for pattern recognition. These produce a value between 0 and 1, where 1 denotes a perfect match in our case.

Figure 9 shows an example of an ANN that can be trained to classify M different device or device types using signatures having N bins. This is a multi-layer feedforward ANN which consists of an input layer, a hidden layer and an output layer. In this case, the input layer takes in a vector of size N (b_1 to b_N), and produces an output vector of size M (d_1 to d_M). The elements of the input vector correspond to the values in the probability distribution (signature) and the elements of the output vector correspond to the *similarity measure* between the input signature and the M device or device type signatures that it was trained on. The number of hidden nodes P, that provide optimum results was emperically determined to be 50. Two neural networks of this kind are used for each traffic type that is analyzed. One is trained for device identification while the other is trained for device type identification. Once trained, the neural networks (Θ) are stored in a master database for future use.

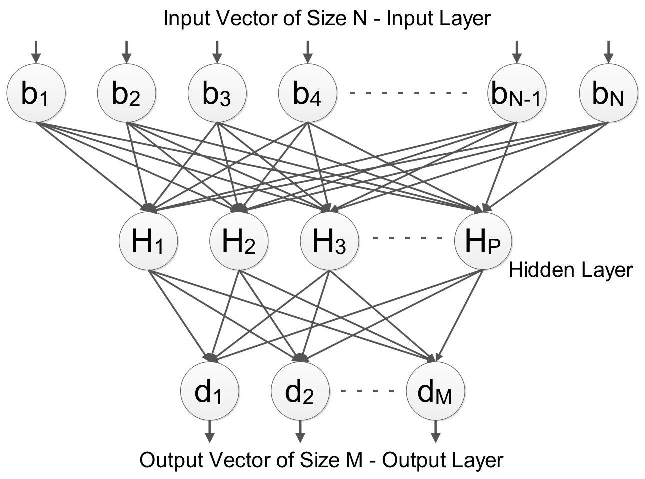


Figure 9: Sample Neural Network

4.3.2 Classification

As explained earlier, GTID compares a device in question with previously collected master signatures and identifies the device in question and/or its type. The successful identification of an unknown device with one of the master devices is refered as *Device Identification* and the identification of the unknown device's type with one of the master device types is refered as *Device Type Identification*. For instance, GTID may have a collection of master signatures for two identical Kindles. In this case, there will be two master device IDs (Kindle#1 and Kindle#2) and one device type (eReader). Hence, given a set of master signatures, there would be three applicable outcomes in identifying a device and its type.

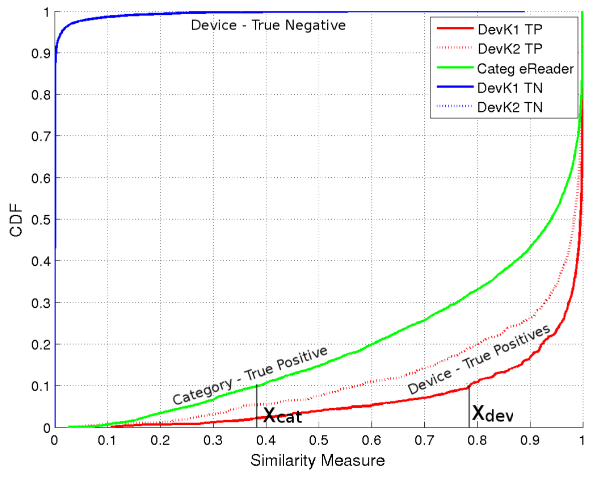


Figure 10: CDF plot of closeness value for kindle fire - TCP traffic

In the first one, GTID successfully recognizes the unknown device and its device type because the samples from the unknown device match either one of the master signatures of a device or master signatures of a device type in the signature database. In the second one, GTID is not able to find a match for a given device and device type in the signature database. Therefore, in this case, the sample device is classified as an unknown device. The third outcome represents a case between the first two outcomes as the system is able to identify the device's type, but not the actual device associated with the tested device.

GTID's core algorithm is shown in Algorithm 1. Based on the type of traffic (Table 3), the system extracts the neural networks (Θ_{ID}, Θ_{Type}) and lists (Dev_{list}, $Type_{list}$) from the master database (line 3). Masking vectors are then generated according to the subset of master signatures that is used for comparison (line 4). The system then extracts the unknown signature Ω from the unknown sample S (lines $5-6$). Once this is extracted, the system feeds it into the device ID neural network (Θ_{ID}). The masked output generated by the neural network is then used to get the *index* and the corresponding value of *closeness* (lines $7-8$). The closeness values range between 0 to 1, where 1 denotes a perfect match. If the value of closeness fits the previously observed True-Positive (TP) closeness values (X), the unknown signature is identified as the device pointed by the *index* and its corresponding type (lines $9-11$). If not, similar steps are performed to get the *index* and *closeness* value for device type using Θ_{Type} (lines $12-14$). If the device type closeness satisfies the condition in line 16, the system identifies the unknown signature to be from an *Unknown* device and a known category pointed by the *index*, else the signature is identified to be from an *Unknown* device and an *Unknown* type (lines $15-19$). Note that the TP values used to determine X come from a database of TP values which was created using a separate dataset. Thus, GTID checks to see if the closeness value fits the previously seen TP distribution of the master signature in order to determine whether a signature is classified as known or unknown. An example distrubution of TP history for Kindle (device) and eReaders (device type), along with the cutoff TPs X_{dev} and X_{cat} is shown in Figure 10. From the Kindle and eReader TP distributions in Figure 10, it can be observed that device (Kindle) TPs (red line) are more often closer to 1, compared to the device type (eReader) TPs (green line).

The clearly observed difference in the distribution patterns of device TP, the device type TP and TN from other devices (Figure 10) is in fact due to heterogeneity of the different hardware composition (e.g., processor, DMA controller, memory) of the devices as well as clock skew and possibly the intrinsic variation in the chip fabrication process. Therefore, a tested device is first expected to be closest to its own signature, next to its type signature, then to other devices, assuming the existence of a match for the signature of the tested device.

4.3.3 Metrics for GTID's Effectiveness

The performance of GTID is evaluated using *accuracy* and *recall* as our metrics similar to [35]. Accuracy is defined as:

$$\alpha = \frac{TP + TN}{TP + TN + FP + FN}, \tag{2}$$

where *TP, TN, FP*, and *FN* refer to True Positive, True Negative, False Positive, and False Negative, respectively. The analysis uses accuracy to measure the overall performance of the proposed system. Recall is the measure of identifying an actual device and it is statistically defined as:

$$\gamma = \frac{TP}{TP + FN}. \tag{3}$$

The analysis use both accuracy and recall because the sole usage of accuracy is misleading when analyzing certain types of test cases (e.g., for test cases that do not allow the entire cohort to contribute to all of the statistics). This is because accuracy, as shown in Equation 2, requires statistics from the entire cohort of devices (i.e., TNs). This information may not be available for certain experiments (i.e., different protocols on one device). Hence, recall makes the evaluation independent of the impact of the other TNs and yields a realistic performance focused only on TPs. Nonetheless, accuracy is still useful, for instance, in analyzing the behavior across different traffic types of the entire cohort. Thus, in the Performance Evaluation Section, accuracy is only populated where appropriate in the results.

Table 1: Isolated Network Device Specifications

Device	Device ID	Model	Hardware Specification	Operating System	Kernel
Netbook	Dell 1	DELL Latitude 2110	Intel Atom N470 @ 1.83GHz 1GB RAM	Ubuntu 10.04.1 LTS /Windows XP	Kernel 2.6.32-24 -generic
	Dell 2				
	Dell 3				
	Dell 4				
	Dell 5				
Nokia	Nokia 1	N900	ARMv7 rev 3 (v7l) @ 600MHz 256MB RAM	Maemo 5, Version 3.2010.02-8	Kernel 2.6.28 - omapl
	Nokia 2				
iPhone3G	iPhone3G 1	MB715LL (A1303)	A4 processor @ 1GHz 512MB eDRAM	iOS 4.0 (8A293)	Kernel 10.3.1
	iPhone3G 2				
iPhone4G	iPhone4G 1	MC608LL (A1332)	A4 processor @ 1GHz 512MB eDRAM	iOS 4.3.3 (8J2)	Kernel 11.0.0 /Firmware 04.10.01
	iPhone4G 2				
iPad	iPad 1	MC497LL	A4 processor @ 1GHz 256MB DDR RAM	iOS 4.3.5	Kernel 10.3.1
	iPad 2			iOS 3.2.2	
	iPad 3			iOS 3.2.2	

Table 2: Campus Network Device Specifications

Device	Device ID	Model	Hardware Specification	Operating System	Kernel
Asus Netbook	AS1	Asus EeePC 1025C	1.6 GHz Intel Atom processor-N2600 1GB RAM	Ubuntu 12.04 (32 bit) / Windows 7 (32bit)	Linux 3.2.0-29 - generic
	AS2				
	AS3				
	AS4				
	AS5				
Lenovo	L1	Lenovo G570	2.4 GHz Intel Core i5-2430M 4GB RAM	Ubuntu 11.04 (64 bit) /Windows 7 (64 bit)	Linux 2.6.38 - 13 - generic
	L2				
Dell	D1	Dell Probook 4350s	2.4 GHz Intel Core i5-2430M 4GB RAM	Ubuntu 11.04 (64 bit) /Windows 7 (64 bit)	Linux 2.6.38 - 13 - generic
	D2				
ASUS Tablet	T1	ASUS Transformer TF 101	1.0GHz NVIDIA Tegra 2 dual-core CPU 1GB RAM	Android 3.2.1	Kernel 2.6.36.3
	T2				
Google Nexus One	G1	Nexus One	1 GHz Qualcomm QSD8250 Processor 512MB RAM	Android 2.2	Kernel 2.6.32.9
	G2				Kernel 2.6.29
Kindle Fire	K1	Kindle Fire	1 GHz Texas Instruments OMAP 4430 dual-core processor; 512MB RAM	Customized Android 2.3	Firmware 6.2.2
	K2				Firmware 6.2.1
Apple TV	A1	ATV 1st Gen	Intel Pentium M processor 256MB DDR2 RAM @400MHz	OS Version 2.0 based on Mac OS X	-
	A2				
HP Printer	H1	HP Officejet 6500A Plus	-	RTOS	-
	H2				
D-Link IP-Camera	C1	D-Link DSC 932L	-	RTOS	-
	C2				
PS3	P1	CECH-3001A	CPU : Cell Processor PowerPC-base Core @3.2GHz. GPU: RSX @550MHz 256MB XDR Main RAM @3.2GHz 256MB GDDR3 VRAM @700MHz	XrossMediaBar	Firmware Version 3.72
	P2				

4.4 Performance Evaluation

This section evaluates the performance of GTID across three dimensions. First, the technique is analyzed in an isolated network environment (Figure 11). Traffic captures on this setup was performed by the authors of [1]. The pcap files from their experimentation was borrowed for analysis of the newly proposed technique. Second, the performance of GTID is measured in a live campus network (Figure 13) during peak hours in both wireless and wired setups. Finally, the effectiveness of GTID is studied under various attack scenarios.

4.4.1 Setup of Testbeds

Two automated testbeds were assembled to transmit and record traffic from the wireless devices to the wired segment and vice versa. In the isolated testbed setup by the authors

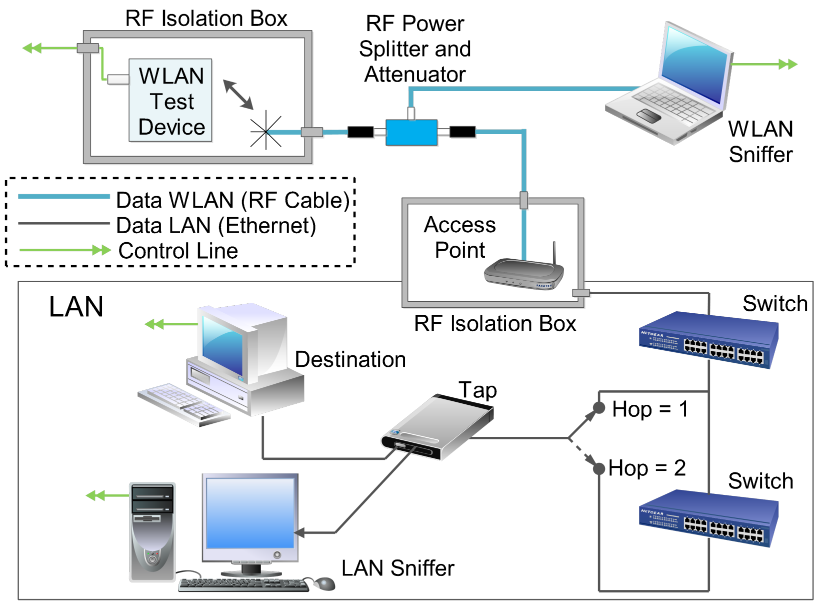


Figure 11: APL testbed setup

of [1] (Figure 11), a control machine (not shown in the figure) was used to send commands to the different devices in the testbed for single and multiple hops scenarios. The device under test was placed in an isolation box to reduce RF leakage and interference. For the real network testbed (Figure 13), the AP and LAN destination were connected to a campus backbone switch to test the device with real MAC and physical layer interference from other wireless users in proximity, during peak hours. The same experiments were repeated using the wired interface on devices, by connecting them directly into the back bone switch. A total of 37 different devices, (Figure 12) were tested and the details of which are listed in Tables 1 and 2.

4.4.2 Traffic & Signature Generation

Two generic applications were used to generate traffic in our testbeds. One was *Iperf*, which was used to generate both TCP and UDP traffic at controlled rates, and the other was *Ping*. The analysis also includes *Active Fingerprinting* for which IATs of ping responses were used for characterizing a device. In addition to these, tests were performed using day to day applications such as secure copy (*SCP*) and *Skype*. Table 3 presents the list of traffic types and the associated parameters that were used for performance analysis.

Figure 12: Devices used

The analysis uses one hour of traffic from each of these traffic types. The captures for the devices listed in Table 1 was provided by the authors of [1], while the rest was captured by us using the automated testbed shown in Figure 13. In addition to these, a few wireside captures were also made to show that GTID can be applied for both wired and wireless environments. This produces a total of 400+ hours (more than 500 gigabytes) of traffic which was used to evaluate GTID. The first half of each capture is used for training the ANN and the second half is used for performance analysis. The best sample size for a good *overall* performance of GTID was empirically determined to be 2.5K packets.

GTID operates in two modes: *Known* and *Unknown*. The known mode refers to a case where GTID attempts to recognize a previously seen device among other *previously seen*

Table 3: Traffic Types Used in Experiments

Exp #	Active/Passive	Traffic Type	Traffic Case #	Parameters		
1	Passive	Iperf - UDP	1	-b 1M	-t 3600	-l 56
2			2	-b 1M	-t 3600	-l 1400
3			3	-b 8M	-t 3600	-l 56
4			4	-b 8M	-t 3600	-l 1400
5		Iperf - TCP	1	-t 3600		
6		SCP	1	1.7 GB		
7		Skype - UDP	1	Video		
8		Ping	1	-s 56	-0.01	-c 360000
9			2	-s 1400	-i 0.01	-c 360000
10	Active	Ping Response	1	-s 56	-i 0.01	-c 360000
11			2	-s 1400	-i 0.01	-c 360000

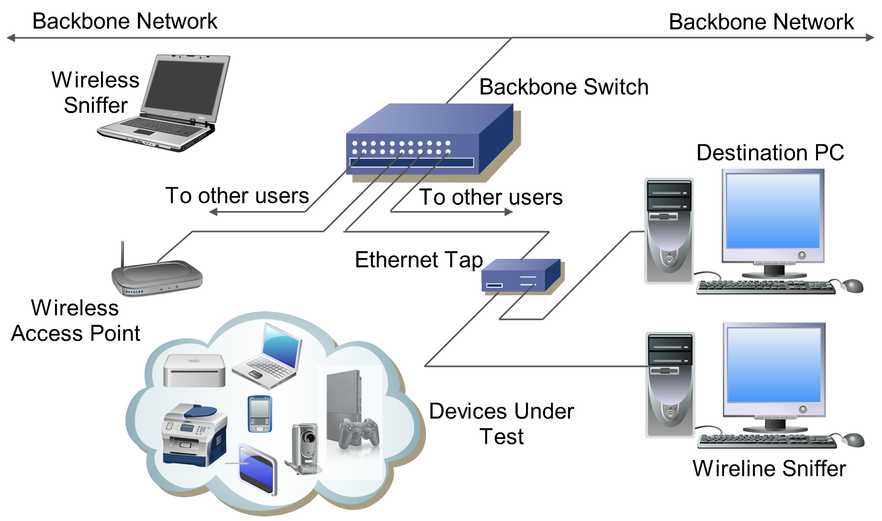


Figure 13: Campus testbed setup

devices and therefore, has a master signature associated with the device in question. Hence, in this case, GTID either correctly identifies the device and the device type (category) or mis-identifies them. In the *unknown* mode of test, GTID is exposed to both devices it has *previously seen* and devices that it has *not previously seen*. Hence, in this case, GTID does not have the necessary master signature associated with a sample device tested. As a result, if GTID does not recognize a device, it then identifies it as an unknown device, otherwise it identifies the type and/or the device. The two different modes work best for different scenarios. For example, in a benign network, the *known* mode, which has better accuracy and recall, can be used for inventory control. However, in a network where access control is a concern, GTID can be employed in *unknown* mode. Using the accuracy (α) and recall (γ) metrics explained in Section 4.3.3, the overall effectiveness of our technique is analyzed for these different modes. Nonetheless, as noted earlier, the performance analysis specific to device IDs and device types, only focuses on γ, because α is inflated superfluously by TNs. Note that this is not needed for results per traffic type because all traffic from devices and types are aggregated for each traffic type; hence, results pertinent to both metrics are included.

4.4.3 Experimental Results from Isolated Testbed - Wireless

Table 4: Isolated Network Testbed (Passive Analysis)

	Device ID					**Device Type**				
		Known		**Unknown**			**Known**		**Unknown**	
	Dev	α	γ	α	γ	**Type**	α	γ	α	γ
Max	*Netbook #5*	-	1.00	-	0.93	*iPhone4*	-	0.99	-	0.89
Min	*Iphone4 #2*	-	0.74	-	0.66	*iPhone3G*	-	0.15	-	0.18
	Traffic Type					**Traffic Type**				
Max	*UDP 1400B 1Mbps*	1.00	0.97	0.98	0.82	*UDP 1400B 1Mbps*	0.87	0.66	0.87	0.61
Min	*SCP*	0.99	0.89	0.97	0.80	*SCP*	0.87	0.67	0.87	0.61
Avg		0.99	0.94	0.97	0.81		0.87	0.67	0.88	0.64

The captures generated by the authors of [1] were used to test the performance of GTID in an isolated network environment (Figure 11). Conducting experiments in an isolated network allowed for fundamental and deeper understanding of the overall technique. Specifically, we seek to understand: *(1) What is the overall accuracy and recall of this technique in an isolated environment? (2) Is there a protocol/rate that works the best for this technique in the testbed? (3) Are there devices that are more amenable to this technique? (4) How does data rate affect this technique overall?* A summary of results for this analysis is shown in Table 4. For more detailed results, please refer to Section A.1.

In a single hop case, as seen in Table 4, the device with the maximum γ is *Netbook #5* with 100% while the device with the minimum γ is *iPhone4 #2* with 74% and the average is 94% for the device identification analysis in the known analysis mode. For the same devices, in the unknown mode, the maximum, minimum, and the average fall to 93%, 66%, and 81%. The maximum, minimum, and the average recall values for both known and unknown test modes for device types identification are lower compared to that of device identification. The average recall values are 67% and 64% for the known and unknown test modes, respectively. One reason for this is lack of enough representative devices to train for device type identification. Since most device types have only two devices (Table 2), we had to use one device for training (which wasn't sufficient) and the other for performance evaluation. It is shown through experiments in Section 4.4.4 that recall of device type identification can be significantly improved by using more number of training samples (Figure 14). As for the protocols and applications tested, UDP with the rate of 1Mbps shows a recall of 97% (known) and 82% (unknown) for device identification and 66% (known) and 61% (unknown) for device type identification. The minimum for this four-tuple, device identification (known: 89% vs. unknown: 80%) and device type identification (known: 67% vs. unknown: 61%) belongs to the SCP application. The slight performance penalty with SCP, in comparison with UDP, is attributed to the additional processing time that is specific to the application layer (e.g., encryption).

4.4.4 Experimental Results from Campus Network - Wireless

Both active and passive experiments were conducted on a live network to determine the feasibility of the technique and to provide bounds on the performance of GTID in realistic deployments. The real network analysis was specifically meant to answer the following set of question: *What is the overall accuracy (α) and recall (γ) of this technique in a real network?* and *Is there a traffic type or device that is more amenable to this technique?* The results of the active and passive analysis for the campus network analysis are summarized in Table 5 and Table 6 respectively. For detailed results, please refer to Section A.2

Table 5: Campus Network - Wireless (Passive Analysis)

	Device ID					**Device Type**				
		Known		**Unknown**			**Known**		**Unknown**	
	Device	α	γ	α	γ	**Device Type**	α	γ	α	γ
Max	*Kindle #2*	-	0.93	-	0.85	*Asus Netbook*	-	0.99	-	0.87
Min	*Tablet #2*	-	0.48	-	0.41	*Google Ph*	-	0.22	-	0.30
	Test Type					**Test Type**				
Max	*UDP 1400B 8Mbps*	0.96	0.80	0.95	0.70	*Skype*	0.96	0.96	0.95	0.90
Min	*TCP*	0.95	0.74	0.94	0.56	*UDP 1400B 8Mbps*	0.77	0.54	0.80	0.52
Avg		0.95	0.74	0.94	0.64		0.86	0.68	0.87	0.63

Passive Analysis : As seen in Table 5, in the campus network testbed the device with the maximum γ is *Kindle #2* with 93% while the device with the minimum γ is *Asus Tablet #2* with 48% and the average is 74% for the device identification analysis in the known operational mode. In the unknown mode of these devices, the maximum, minimum, and the average γ fall to 85%, 41%, and 64%, respectively. Similar to what was observed in the isolated experiments, the maximum, minimum, and the average recall values for both known and unknown test modes for device type identification is less compared to device identification. The average values of recall are 68% and 63% for the known and unknown test modes, respectively. As mentioned in Section 4.4.3, the performance of device type identification can be significantly improved by using more number of training samples for each type of device. Figure 14 shows the results of an experiment that was conducted to see if our hypothesis was true.

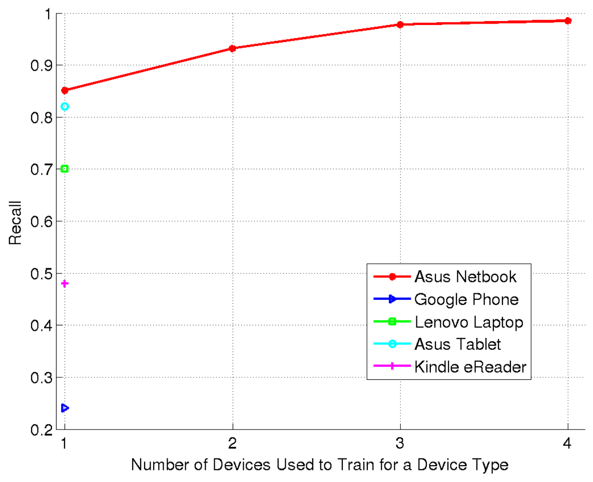


Figure 14: Effect of Increasing Training Data

This particular experiment starts by training on one representative device for each device type, and continues to increase in steps of 1, the number of representative devices used to train the Asus netbooks (device type). From the results (Figure 14), the recall of the Asus netbook (device type) is clearly observed to increase as the number of training samples is increased. As for the applications and protocols, maximum and minimum are different for device identification and device type tests. Specifically, for the device identification tests, UDP with a rate of 8Mbps and 1400B payload exhibits the maximum γ: 80% while for the device type tests, the maximum is achieved by Skype traffic with γ: 96%. Further results on the recall for all traffic types and devices/device types are shown in Figure 15 and Figure 16 respectively. In the future, we plan to determine why the recall values of these signatures are traffic and device type dependent.

By comparing the results of the real network to the isolated network, one can observe

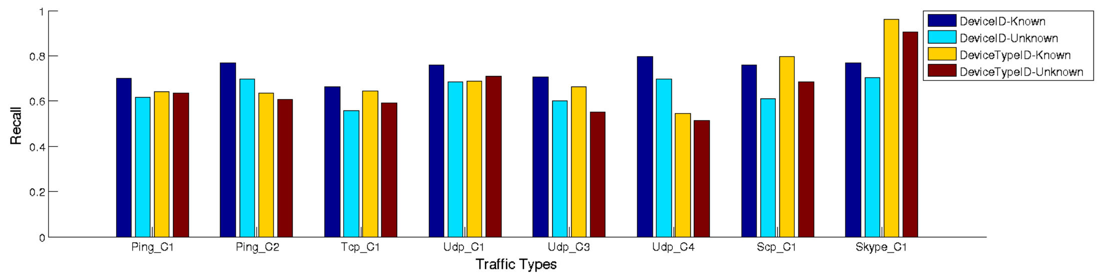


Figure 15: Recall vs Traffic Types

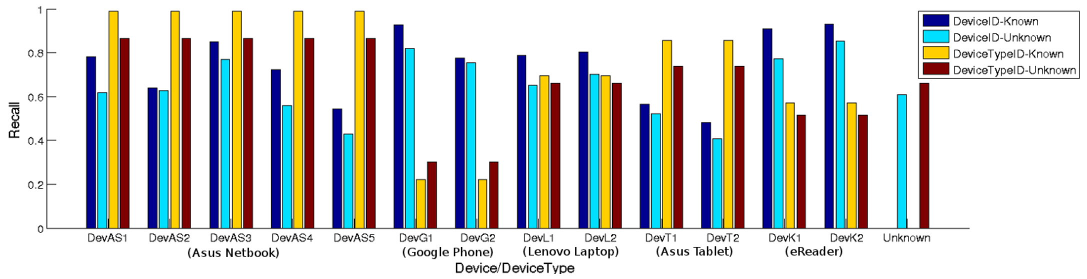


Figure 16: Recall vs Devices/DeviceTypes

the degradation in performance across both α and γ in the real network. It is also observed that the device identification results are better than the device type identification, across all the cases. This can be attributed to the lack of enough devices to train for each device type.

Results from the analysis of the impact of physical and MAC layer interference on the real network is presented in Figure 17. Focusing on the similarity measures for these two testbeds, it is seen that more than 50% of the similarity measures have values close to 1 (which denotes a perfect match) in the isolated testbed, while its less than 30% for the real testbed. This difference is attributed to the uncontrolled characteristics of the physical medium which make inter arrival patterns look less similar in the real network.

Finally, the slight performance decrease associated with the unknown test mode observed in all the test scenarios (Figure 15) is attributed to the nature of the identification algorithm. However, our binary identification technique utilizing TPs, FPs, TNs, and FNs, shows strong promise for the effectiveness of GTID (given its numerous benefits).

Active Analysis : In this analysis, ping requests were used as probes to trigger responses (ping response) from the device. Using this method, a network administrator will be able

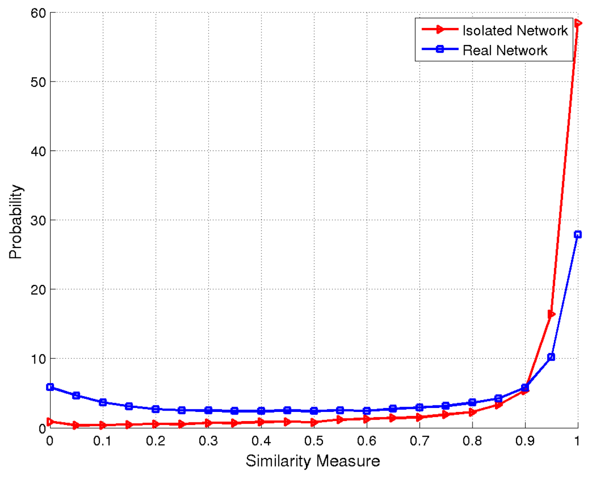


Figure 17: Effect of MAC Contention

to fingerprint any device that responds to ping (e.g., IP camera). Due to this advantage, we were able to fingerprint a total of 23 devices using this technique. The list includes devices such as play station, apple TV, IP camera, HP printer, etc. Ping requests of two different payload sizes (Table 3) were sent at 10ms interval to generate a continuous stream of responses from the targeted device. The IATs from the response stream were then analyzed using GTID. The results of this analysis are summarized in Table 6.

Table 6: Campus Network - Wireless (Active Analysis)

	Device ID					**Device Type**				
		Known		**Unknown**			**Known**		**Unknown**	
	Device	α	γ	α	γ	**Device Type**	α	γ	α	γ
Max	*Kindle #2*	-	0.91	-	0.85	*Printer*	-	1.00	-	0.91
Min	*Printer #2*	-	0.35	-	0.08	*Laptop*	-	0.55	-	0.56
	Test Type					**Test Type**				
Max	*Ping 1400B*	0.97	0.72	0.96	0.58	*Ping 1400B*	0.95	0.77	0.94	0.71
Min	*Ping 56B*	0.97	0.66	0.96	0.51	*Ping 56B*	0.94	0.71	0.94	0.70
Avg		0.97	0.69	0.96	0.55		0.94	0.74	0.94	0.70

From the results summary shown in Table 6, it can be observed that *Kindle #2* once again has the highest γ of 91%, which shows that it quite suitable for fingerprinting using GTID. The minimum recall was shown by the HP printer with 35% recall in the known mode of operation. However, the same printer has the highest value of recall for device type identification. This shows that the printers possess signatures that are very different from other types of devices, but quite similar to each other. From the per traffic type analysis, it can be seen that Ping with 1400Byte payload does better than Ping with a 56Byte payload.

This holds good for both device and device type identification and also across both known and unknown analysis. This we believe is due to the fact that a continuous stream of large packets will involve more memory operations, which in turn will provide a stronger fingerprint of the device's hardware. In general, the active mode of fingerprinting performs slightly worse compared to the passive analysis. This is because, in active fingerprinting, the IAT patterns embed the signature of both the probing device and the target device. Looking into methods for removing the sender's signature from the IAT pattern is a part of the proposed future work.

4.4.5 Experimental Results from Campus Network - Wireline

This section briefly shows that GTID can be easily extended to a wired environment and thus is not limited to a wireless setup. In this setup, the target devices are directly wired into the backbone switch instead of connecting through an AP as shown in Figure 13. Similar to the analysis in Section 4.4.4, both active and passive traffic types are used to evaluate the performance of GTID in a wired environment. A summary of results obtained from this analysis is shown in Table 7 and 8. For more detailed results, please refer to Section A.3.

Table 7: Campus Network - Wireline (Passive Analysis)

	Device ID					**Device Type**				
		Known		**Unknown**			**Known**		**Unknown**	
	Device	α	γ	α	γ	**Device Type**	α	γ	α	γ
Max	*Dell #1*	-	1.0	-	0.88	*AsusNetbook*	-	1.00	-	0.84
Min	*AsusNetbook #2*	-	0.39	-	0.35	*Dell*	-	0.26	-	0.41
	Test Type					**Test Type**				
Max	*Ping 56B*	0.97	0.86	0.94	0.72	*TCP*	0.91	0.87	0.96	0.90
Min	*UDP 56B 1Mbps*	0.89	0.50	0.88	0.42	*UDP 1400B 1Mbps*	0.66	0.49	0.75	0.51
Avg		0.92	0.65	0.92	0.68		0.92	0.57	0.84	0.67

Passive Analysis: The summary of results in Table 7 shows that GTID is feasible on a wired network as well. Dell #1 shows the highest value of recall for device identification while Asus netbooks show the best recall for device type identification. One major reason for this is that, Asus netbooks had many more devices to train for the device type, while the rest of the devices had just one (refer Figure 14). While one would expect the performance of GTID to be better on a wired network as compared to a wireless network (no physical or MAC layer contention), this is not what was observed. The results of the wired network analysis show slight degradation in the performance of GTID in comparison to the results obtained from the wireless setup. The reasons for this could be 1) use of different sets of devices for analysis in the wired and wireless analysis; and 2) binwidths for a wired network might have to be configured differently due to lesser timing fluctuations and higher rates. A more detailed analysis of this will be done as a part of the future work. With respect to traffic type analysis, we see that that TCP is performing the best with a recall of 90%. However, in the wireless analysis, TCP performed poorly (recall of 59%) when compared to other traffic types (refer Table 20 in Appendix). This shows the impact of delays and retransmissions on a wireless network which directly impacts the behavior of TCP flows. The absence of such delays and packet losses on wired networks produced more consistent IAT patterns and thus better signatures for TCP. It is also seen that 1400 byte UDP traffic at 1Mbps has the lowest recall for both the wired and wireless analysis. This traffic type is one of the slowest in our analysis with average inter arrival times of 11.2ms. This shows that GTID works better with traffic that have higher packet rate (i.e., smaller IAT).

Table 8: Campus Network - Wireline (Active Analysis)

	Device ID					**Device Type**				
		Known		**Unknown**			**Known**		**Unknown**	
	Device	α	γ	α	γ	**Device Type**	α	γ	α	γ
Max	*PlayStation #2*	-	0.98	-	0.91	*Apple TV*	-	1.00	-	0.90
Min	*AsusNetbook #5*	-	0.21	-	0.45	*Dell*	-	0.08	-	0.05
	Test Type					**Test Type**				
Max	*Ping 1400B*	0.96	0.67	0.96	0.61	*Ping 56B*	0.96	0.87	0.94	0.75
Min	*Ping 56B*	0.96	0.66	0.94	0.49	*Ping 1400B*	0.95	0.82	0.94	0.77
Avg		0.96	0.66	0.95	0.55		0.96	0.85	0.94	0.76

Active Analysis : Table 8 shows a summary of results obtained for Active analysis on a wired network. It can be observed that PlayStation#2 shows the best performance for device identification while the AsusNetbook#5 shows the worst. This might be partially due to the fact that there are more AsusNetbooks in the list of devices compared to the rest which have only two. For device type identification, *AppleTV* has the highest recall (100%) while Dell has the lowest. It is easy to see from Table, 7 and 8 that the Dell device type is the most difficult to classify. After further investigation, it was found that the Dells had a few differences in the hardware (including the NICs) even though they were sold with the same model number. This, in a way shows the sensitivity of GTID to hardware differences/similarities.

4.5 Attacker Models

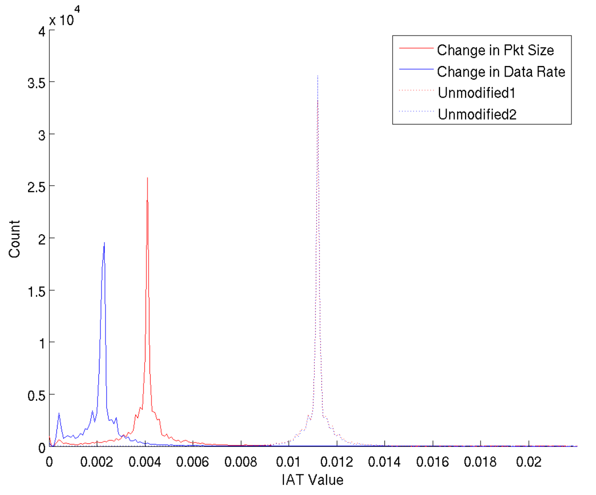


Figure 18: PDF of UDP IATs for Normal and Attacker Traffic

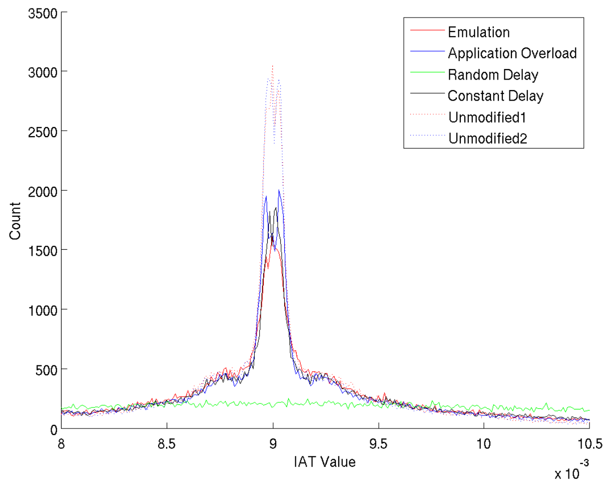


Figure 19: PDF of Ping Response IATs for Normal and Attacker Traffic

This section seeks to determine the effectiveness of GTID under a number of attack scenarios. This analysis assumes that the attacker has some knowledge of the detection technique and is capable of controlling his device's network traffic. Given that this technique is IAT-based, a *novice attacker* would be prompted to do one of the following: (1) *Introduce constant/random delays to packet stream*; (2) *Vary the packet size*; (3) *Modify/change the operating system*; (4) *Load the CPU with intensive applications to over shadow normal behavior*; (5) *Tunnel packets through another protocol.* These attacks were carried out in an attempt to evade GTID. In the scenario, we assume all the devices are *known.* GTID detects these attacks and classifies all of these devices that generated attack traffic from previously seen devices as *unknown*, which is a red flag to a network administrator. The variation in the IAT distribution patterns (from normal) observed in Figure 18, 20, 19, and 21 explains why GTID was able to identify the attacker traffic.

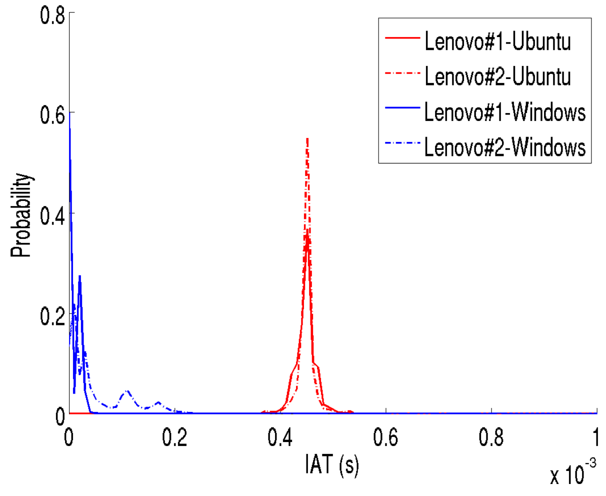


Figure 20: O/S Attack

However, *if the attacker is highly skilled* and knowledgeable of the technique, she could try to emulate an authorized device in order to establish/maintain network access. To do this, the attacker would need the distribution of the *difference* in the IATs of her device and the device that she wants to emulate. Once she has this information, she can feed it into a network emulation tool like *netem* (which is available in linux kernels 2.6 and higher) to transmit packets in accordance with the distribution. When such an attack is performed, one would expect the attacker's device to be classified as a known device. However, as seen in Figure 21 this is not true. Figure 21 shows the IAT distribution when the Lenovo laptop attempted to behave like a Kindle. Clearly, the distribution of the emulated traffic is different from the actual and the targeted device and GTID labels this traffic as unknown. Also, the IATs are observed to be more distributed when compared to the unaltered device. One of the primary factors that prevented an accurate emulation is the fact that the attacker's device has to simultaneously spoof a signature of a device and attempt to hide its innate signature. This is because, the variation in the IATs that the device attempts replicate gets added to its innate timing variation with results in a signature that is different from both the device that is trying to emulate and the device that is being emulated. It is important to note that the theory behind GTID is that different devices essentially "talk" differently (i.e., they have a different cadence), so as illustrated above, it is difficult for even a more powerful device to emulate the traffic distribution of a less powerful device.

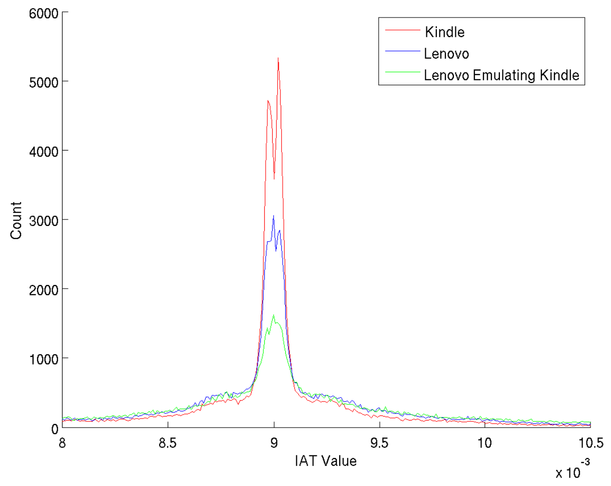


Figure 21: Laptop Emulates a Kindle

4.6 Sub-module Architecture

Detection techniques need to be quick in order to be of any practical use. High accuracy with high latency may lead to correct, but less useful results. In order to study the detection speed of this technique, a prototype version of GTID was developed in MATLAB and its performance was tested. The internal architecture of the prototype is shown in Figure 22.

4.6.1 Overview

The basic purpose of this tool is to match a sample from a given network traffic with a list of previously stored signatures. This is achieved using a three stage process as explained below.

Stage1 : Network Traffic Collection: Before starting to collect traffic from the network interface, the tool takes in necessary parameters such as Sample Size (S), Number of Samples (N), and MAC address of the suspicious device. *Tcpdump* is then used at the back end to collect and save N number of pcap files, each containing captures of $S+1$ packet headers. These files are sent to stage two (identification) of the process as and when they are captured. This results in a pipeline which thereby contributes to the speed up GTID's detection time. Another alternative that the tool supports is taking existing pcap files as input. When this option is used, the tool parses the given pcap file in stage one and sends N IAT matrices of size S to stage two of the process. In this case, stages one and two cannot be pipelined and therefore, the time required to complete stage one is highly dependent on the size of the pcap file. Analysis of preexisting captures is very handy in forensics.

Stage2 : Identification: At this stage, once a pcap sample arrives, IAT values are extracted using [36]. After extraction, the IAT values are passed to the identification module which is the core module of this entire process. This module is responsible for matching the given IAT pattern to a preexisting pattern in the master database. After processing, it gives out the identified device/type along with the similarity measures. This process of extraction and identification can be done in parallel because each incoming pcap file is a separate sample. In order to parallelize this process, the tool assigns each sample to a new thread that runs on a separate core or hyper threads in the CPU. This is done using n matlabpool workers and parallel loops. If this stage receives sampled IAT matrices, then it skips the extraction phase and directly enters into the process of identification as shown in Figure 22. This happens when pcap files are fed into stage one of the process.

Stage3 : Decision Making: The decision making stage is simple and straight forward. It waits and accumulates all the results obtained from the identification module until the Nth sample is identified. Once it has all N results, it checks for the device/type that has

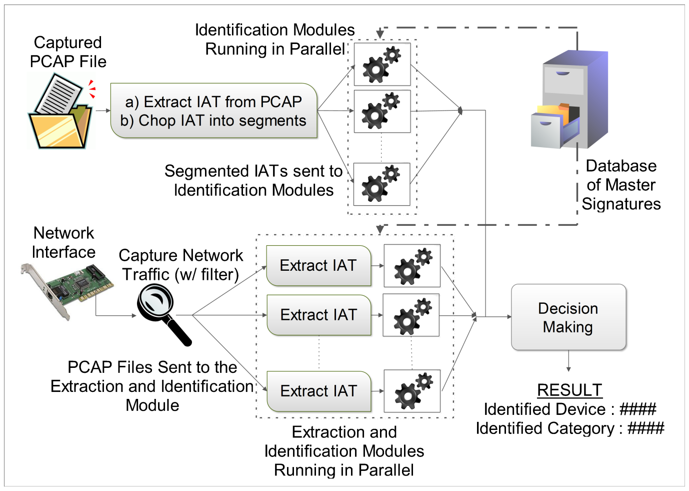

Captured PCAP File
Identification Modules Running in Parallel
a) Extract IAT from PCAP
b) Chop IAT into segments
Segmented IATs sent to Identification Modules
Database of Master Signatures
Network Interface
Capture Network Traffic (w/ filter)
Extract IAT
Extract IAT
Extract IAT
PCAP Files Sent to the Extraction and Identification Module
Decision Making
RESULT
Identified Device : ####
Identified Category : ####
Extraction and Identification Modules Running in Parallel

Figure 22: Real-Time Implementation

matched most of the input samples and declares that device/type as the final identified device/type. The tool uses the graphical user interface (GUI) provided by the proposed framework to take in values and display results. The GUI can be used to set all the parameters that are required for traffic capture and identification. Moreover, it has features which enables the user to select a subset of all the master signatures that is in the database which will in turn help in both speeding up the identification process and also in reducing the number of FPs. This is useful if one has an idea of the type of unknown device that is under test.

4.6.2 Performance Analysis

In order to quantify the performance of this technique, timing analysis was performed on the GTID tool. The two most important factors that delay the identification process are *capture time* and *processing time*. The capture time is in turn dependent on the sample size and the processing time is in turn dependent on the algorithm and processing power. Table 9 shows the variation in the time taken to identify a device when the sample sizes are varied. The time taken to capture increases linearly with an increase in sample size (as expected). However the processing time increases at a slower rate. This is evident from the decrease in the percentage of processing time from 4% to 2%, as sample size is increased from 1K to 50K IATs.

Table 9: Timing analysis for variation in sample sizes : Traffic - UDP 1400byte 8Mbps , Number of samples 20

Sample Size	**Capture Time(s)**	**Processing Time(s)**	**%Processing Time**
1K	1.18	0.051	4.14
2.5K	3.95	0.12	2.95
5K	7.59	0.199	2.55
7.5K	11.28	0.28	2.42
10K	14.98	0.38	2.47
30K	44.27	1.15	2.53
50K	73.61	1.18	1.58

Table 10: Effect of Parallelizing the Process : Sample Size 2500, Number of Samples 20

Test Type	**Total Processing Time(s)**	
	1 Thread	**8 Thread**
UDP 56B 1Mbps	30.66	26.24
UDP 56B 8Mbps	11.49	6.67

Since the capture time is inversely proportional to the packet rate, the percentage of the processing time will not be as low as it appears in Table 9. At high data rates, the processing time will start to become the performance bottle neck. The only way to overcome this bottle neck is by parallelizing the identification process. Table 10 shows the time saved by the tool when it was run over a single core and over a hyper threaded quad-core CPU (Intel Corei7). Clearly, the process of parallelizing pays off when the packet rate increases, or in other words, when the processing time becomes comparable to that of the capture time. For the second test case, the time taken for identification reduced by almost 45% when parallelized.

CHAPTER V

CONCLUSION AND FUTURE WORK

This work introduces a framework for fingerprinting networked systems. The design consists of three modules that are independent from one another, thus enabling easy modification/improvements in the future. Such a design enables easy addition/removal of network interfaces, network monitoring tools and fingerprinting techniques. The framework integrates the individual fingerprinting modules using a centralized decision engine which makes the process of fingerprinting more robust. At present, the framework supports a limited number of interfaces and fingerprinting techniques. However, in the future, this will be extended to include techniques such as clock skew identification, AP fingerprinting, protocol fingerprinting, etc. Currently, the decision module provides users the control over customizing the weights for a particular technique. In the future, the framework will be improved to auto detect the climate of the targeted network to adjust the weights accordingly.

This work also introduces GTID, a passive technique for device and device type fingerprinting. GTID exploits the heterogeneity of devices, which is a function of the different device hardware compositions and the inherent variation clock skews. This technique was applied to the challenging problem of access control in 802.11 networks. The effectiveness and the practicality of GTID was demonstrated on both an isolated testbed and a live campus network using artificial neural networks. Further, it was shown, using a collection of 37 devices with a diverse set of operating systems, that GTID had high accuracy and good recall in identifying previously seen and unknown devices and device types. This work also addresses the efficacy of GTID under various attack models and considered the performance of a real-time implementation of GTID. In the future, this work will be extended to understand the efficacy of GTID when device and device type identification is conducted over various access links (e.g., DSL, LTE). Additionally, the robustness of GTID will be improved to better handle congestion on links and variation in levels of load on a node. Further, the application of GTID will be extended to detecting, counterfeit devices, resource utilization on a node, devices behind a NAT, virtual machines and malware activity. Finally, analysis portion will be extended significantly to address the attacker models with experiments and statistical results.

APPENDIX A

DETAILED EXPERIMENTAL RESULTS

This Appendix section presents detailed results of all experiments that were summarized in Sections 4.4.3, 4.4.4 and 4.4.5.

A.1 Results from Isolated Wireless Experiments

A.1.1 Known Analysis

Table 11: Device ID - Known Analysis Results in Isolated Experiment (Passive)

	Ping_C1		**Ping_C2**		**Udp_C1**		**UdpC2**		**Udp_C3**		**Tcp_C1**		**Scp_C1**		**PerDevice(Avg)**	
	γ	α	γ	α	γ	α	γ	α	γ	α	γ	α	γ	α	γ	α
DevDN1	1.00	1.00	1.00	1.00	0.95	1.00	0.95	1.00	0.99	1.00	1.00	1.00	0.97	0.99	0.98	1.00
DevDN2	1.00	1.00	1.00	1.00	0.88	0.99	1.00	1.00	0.99	1.00	0.98	0.98	0.96	0.98	0.97	0.99
DevDN3	1.00	1.00	0.98	1.00	1.00	0.99	1.00	1.00	0.96	1.00	0.75	0.98	0.74	0.98	0.92	0.99
DevNP1	1.00	1.00	1.00	1.00	1.00	1.00	0.99	1.00	1.00	1.00	0.94	0.99	0.92	0.99	0.98	1.00
DevNP2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.98	1.00	0.88	0.99	0.98	1.00
DevIP1	0.98	1.00	1.00	1.00	1.00	1.00	0.96	1.00	0.95	1.00	1.00	1.00	0.86	0.98	0.96	1.00
DevIP2	0.90	0.99	0.98	1.00	0.92	0.99	1.00	1.00	1.00	1.00	1.00	1.00	0.98	1.00	0.97	1.00
DevIP3	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.98	1.00	1.00	1.00
DevIT1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.69	0.98	0.96	1.00
DevIT2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
DevIF1	0.80	0.93	0.70	0.97	0.97	0.99	0.85	0.98	0.82	0.98	0.65	0.94	0.84	0.94	0.80	0.96
DevIF2	0.37	0.93	0.92	0.97	0.90	0.99	0.83	0.98	0.88	0.98	0.59	0.95	0.69	0.97	0.74	0.96
DevDN4	-NA-	-NA-	-NA-	-NA-	1.00	1.00	1.00	1.00	1.00	1.00	0.99	1.00	1.00	1.00	1.00	1.00
DevDN5	-NA-	-NA-	-NA-	-NA-	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
PerFlow(Avg)	0.92	0.99	0.97	0.99	0.97	1.00	0.97	1.00	0.97	1.00	0.92	0.99	0.89	0.98	0.94	0.99

Table 12: Device Type - Known Analysis Results in Isolated Experiment (Passive)

	Ping_C1		**Ping_C2**		**Udp_C1**		**UdpC2**		**Udp_C3**		**Tcp_C1**		**Scp_C1**		**PerDevice(Avg)**	
	γ	α	γ	α	γ	α	γ	α	γ	α	γ	α	γ	α	γ	α
DellNetbook	1.00	1.00	1.00	0.99	0.72	0.85	0.86	0.96	1.00	0.83	1.00	0.97	0.97	0.76	0.94	0.91
NokiaPhone	1.00	1.00	0.00	0.80	1.00	1.00	1.00	1.00	0.48	0.89	0.85	0.97	0.79	0.95	0.73	0.94
iPad	0.58	0.92	0.67	0.73	0.63	0.72	0.47	0.66	0.67	0.74	0.50	0.70	0.18	0.71	0.53	0.74
iPhone3G	1.00	1.00	0.07	0.81	0.00	0.78	0.00	0.70	0.00	0.80	0.00	0.73	0.00	0.80	0.15	0.80
iPhone4G	1.00	0.92	1.00	0.76	1.00	0.98	1.00	1.00	0.98	0.99	1.00	0.97	0.97	0.94	0.99	0.94
PerFlow (Avg)	0.92	0.97	0.55	0.82	0.67	0.87	0.66	0.87	0.63	0.85	0.67	0.87	0.58	0.83	0.67	0.87

A.1.2 Unknown Analysis

Table 13: Device ID - Unknown Analysis Results in Isolated Experiment (Passive)

	Ping_C1		**Ping_C2**		**Udp_C1**		**UdpC2**		**Udp_C3**		**Tcp_C1**		**Scp_C1**		**PerDevice(Avg)**	
	γ	α	γ	α	γ	α	γ	α	γ	α	γ	α	γ	α	γ	α
DevDN1	0.95	1.00	0.94	1.00	0.65	0.98	0.42	0.96	0.77	0.98	0.86	0.99	0.78	0.98	0.77	0.98
DevDN2	0.95	1.00	0.97	1.00	0.85	0.99	0.91	0.99	0.86	0.99	0.79	0.98	0.93	0.99	0.89	0.99
DevDN3	1.00	1.00	0.18	0.94	0.69	0.98	0.70	0.98	0.64	0.98	0.65	0.98	0.72	0.98	0.65	0.98
DevNP1	0.98	1.00	0.94	1.00	0.80	0.99	0.75	0.98	0.82	0.99	0.76	0.98	0.92	0.97	0.85	0.99
DevNP2	0.85	0.99	1.00	1.00	0.80	0.99	0.98	1.00	0.22	0.95	0.67	0.98	0.45	0.96	0.71	0.98
DevIP1	0.88	0.99	0.91	0.99	0.95	1.00	0.76	0.98	0.80	0.99	0.88	0.99	0.58	0.97	0.82	0.99
DevIP2	0.58	0.97	0.91	0.99	0.64	0.97	0.89	0.99	0.94	1.00	0.92	0.99	0.85	0.99	0.82	0.99
DevIP3	0.93	0.99	0.88	0.99	0.91	0.99	0.90	0.99	0.89	0.99	0.83	0.99	0.86	0.99	0.88	0.99
DevIT1	0.95	1.00	0.85	0.99	0.93	0.99	0.76	0.98	0.82	0.99	0.98	1.00	0.54	0.97	0.83	0.99
DevIT2	0.85	0.99	0.82	0.99	0.93	0.99	0.95	1.00	0.87	0.99	0.90	0.99	0.98	1.00	0.90	0.99
DevIF1	0.69	0.94	0.67	0.97	0.93	0.99	0.80	0.98	0.79	0.98	0.71	0.96	0.83	0.93	0.77	0.96
DevIF2	0.37	0.92	0.86	0.99	0.78	0.99	0.73	0.98	0.84	0.98	0.50	0.95	0.54	0.97	0.66	0.97
DevDN4	-NA-	-NA-	-NA-	-NA-	0.69	0.98	0.81	0.99	0.88	0.99	0.76	0.98	0.80	0.99	0.79	0.99
DevDN5	-NA-	-NA-	-NA-	-NA-	0.95	1.00	0.97	1.00	0.96	1.00	0.93	1.00	0.83	0.99	0.93	0.99
Unknown	0.85	0.89	0.94	0.84	0.97	0.85	0.97	0.84	0.98	0.82	0.87	0.84	0.79	0.85	0.91	0.85
PerFlow(Avg)	0.83	0.97	0.84	0.97	0.83	0.98	0.82	0.98	0.80	0.97	0.80	0.97	0.76	0.97	0.81	0.97

Table 14: Device Type - Unknown Analysis Results in Isolated Experiment (Passive)

	Ping_C1		**Ping_C2**		**Udp_C1**		**UdpC2**		**Udp_C3**		**Tcp_C1**		**Scp_C1**		**PerDevice(Avg)**	
	γ	α	γ	α	γ	α	γ	α	γ	α	γ	α	γ	α	γ	α
DellNetbook	0.94	0.99	0.84	0.97	0.59	0.92	0.76	0.96	0.91	0.94	0.87	0.95	0.89	0.84	0.83	0.94
NokiaPhone	0.93	0.99	0.13	0.85	0.95	0.99	0.90	0.98	0.58	0.83	0.82	0.90	0.55	0.91	0.69	0.92
iPad	0.69	0.95	0.72	0.80	0.73	0.89	0.43	0.73	0.80	0.88	0.47	0.88	0.14	0.73	0.57	0.84
iPhone3G	0.88	0.98	0.37	0.86	0.00	0.81	0.00	0.75	0.00	0.84	0.00	0.74	0.00	0.83	0.18	0.83
iPhone4G	0.95	0.99	0.89	0.96	0.96	0.91	0.91	0.99	0.85	0.98	0.89	0.98	0.78	0.93	0.89	0.96
Unknown	1.00	0.90	0.56	0.73	0.79	0.82	0.65	0.80	0.61	0.79	0.62	0.77	0.50	0.71	0.68	0.79
PerFlow(Avg)	0.90	0.97	0.58	0.86	0.67	0.89	0.61	0.87	0.62	0.87	0.61	0.87	0.48	0.83	0.64	0.88

A.2 Results from Wireless Campus Network

A.2.1 Known Analysis

A.2.1.1 Passive Fingerprinting

Table 15: Device ID - Known Analysis on Wireless Campus Network (Passive)

	Ping_C1		Ping_C2		Tcp_C1		Udp_C1		UdpC3		Udp_C4		Scp_C1		Skype_C1		PerDevice(Avg)	
	γ	α	γ	α	γ	α	γ	α	γ	α	γ	α	γ	α	γ	α	γ	α
DevAS1	0.65	0.96	0.71	0.95	0.93	0.95	0.95	0.98	0.80	0.98	0.91	0.99	0.51	0.96	-NA-	-NA-	0.78	0.97
DevAS2	0.83	0.96	0.58	0.95	0.53	0.94	0.73	0.96	0.70	0.93	0.86	0.97	0.23	0.93	-NA-	-NA-	0.64	0.95
DevAS3	0.70	0.96	0.88	0.97	0.72	0.95	0.88	0.98	1.00	0.99	0.98	0.99	0.78	0.95	-NA-	-NA-	0.85	0.97
DevAS4	0.58	0.90	0.44	0.93	0.74	0.98	0.81	0.94	0.91	0.96	0.88	0.97	0.69	0.95	-NA-	-NA-	0.72	0.95
DevAS5	0.23	0.91	0.77	0.96	0.40	0.92	0.47	0.93	0.23	0.92	0.75	0.96	0.95	0.92	-NA-	-NA-	0.54	0.93
DevG1	1.00	1.00	1.00	1.00	0.62	0.94	0.99	0.99	0.97	1.00	0.95	0.97	0.95	1.00	-NA-	-NA-	0.93	0.99
DevG2	1.00	1.00	1.00	1.00	0.55	0.93	0.99	1.00	0.05	0.90	0.85	0.97	0.99	1.00	-NA-	-NA-	0.78	0.97
DevL1	0.75	0.95	0.28	0.94	0.94	0.98	0.80	0.96	0.89	0.99	0.95	0.98	0.77	0.97	0.92	0.98	0.79	0.97
DevL2	0.60	0.95	0.96	0.94	0.50	0.93	0.70	0.93	0.97	0.99	0.86	0.96	0.83	0.97	1.00	0.98	0.80	0.96
DevT1	0.75	0.91	0.81	0.95	0.28	0.91	0.61	0.93	0.53	0.95	0.31	0.91	0.68	0.94	0.55	0.79	0.57	0.91
DevT2	0.03	0.91	0.58	0.95	0.50	0.92	0.41	0.92	0.67	0.88	0.46	0.91	0.59	0.95	0.60	0.79	0.48	0.90
DevK1	1.00	1.00	0.98	1.00	0.93	0.99	-NA-	-NA-	0.69	0.96	-NA-	-NA-	0.95	0.99	-NA-	-NA-	0.91	0.99
DevK2	1.00	1.00	1.00	1.00	0.95	0.99	-NA-	-NA-	0.78	0.97	-NA-	-NA-	0.93	0.99	-NA-	-NA-	0.93	0.99
PerFlow(Avg)	0.70	0.95	0.77	0.96	0.66	0.95	0.76	0.96	0.71	0.95	0.80	0.96	0.76	0.96	0.77	0.88	0.74	0.95

Table 16: Device Type - Known Analysis on Wireless Campus Network (Passive)

	Ping_C1		Ping_C2		Tcp_C1		Udp_C1		UdpC3		Udp_C4		Scp_C1		Skype_C1		PerDevice(Avg)	
	γ	α	γ	α	γ	α	γ	α	γ	α	γ	α	γ	α	γ	α	γ	α
AsusNetbook	0.97	0.99	0.99	1.00	1.00	1.00	1.00	1.00	0.97	0.92	1.00	1.00	1.00	1.00	-NA-	-NA-	0.99	0.99
GooglePhone	0.00	0.80	0.00	0.80	0.65	0.81	0.25	0.81	0.20	0.77	0.40	0.80	0.04	0.81	-NA-	-NA-	0.22	0.80
Laptop	0.99	0.99	1.00	1.00	0.43	0.73	0.67	0.84	0.56	0.85	0.01	0.67	1.00	0.99	0.92	0.96	0.70	0.88
Tablet	1.00	0.81	1.00	0.80	0.60	0.86	0.84	0.72	0.66	0.79	0.77	0.61	0.99	0.95	1.00	0.96	0.86	0.81
eReader	0.25	0.69	0.20	0.68	0.54	0.89	-NA-	-NA-	0.92	0.98	-NA-	-NA-	0.95	0.85	-NA-	-NA-	0.57	0.82
PerFlow(Avg)	0.64	0.86	0.64	0.85	0.64	0.86	0.69	0.84	0.66	0.86	0.54	0.77	0.80	0.92	0.96	0.96	0.68	0.86

A.2.1.2 Active Fingerprinting

Table 17: Device ID - Known Analysis on Wireless Campus Network (Active)

	Ping_C1		Ping_C2		PerDevice(Avg)	
	γ	α	γ	α	γ	α
DevAS1	0.54	0.97	0.95	0.98	0.74	0.98
DevAS2	0.77	0.95	0.95	1.00	0.86	0.97
DevAS3	0.58	0.95	0.40	0.95	0.49	0.95
DevAS4	0.56	0.96	0.49	0.95	0.53	0.96
DevAS5	0.42	0.96	0.56	0.96	0.49	0.96
DevC1	0.64	0.97	0.49	0.97	0.56	0.97
DevC2	0.49	0.97	0.86	0.98	0.67	0.97
DevP1	0.94	0.96	0.77	0.97	0.85	0.97
DevP2	0.15	0.95	0.54	0.97	0.35	0.96
DevG1	0.86	0.98	0.95	0.99	0.90	0.99
DevG2	0.79	0.98	0.78	0.98	0.79	0.98
DevL1	0.42	0.96	0.73	0.95	0.58	0.95
DevL2	0.92	0.98	0.90	0.99	0.91	0.98
DevH1	0.76	0.97	0.88	0.99	0.82	0.98
DevH2	0.68	0.97	0.77	0.98	0.72	0.98
DevA1	0.64	0.95	0.22	0.96	0.43	0.95
DevA2	0.79	0.98	0.96	0.99	0.88	0.99
DevT1	0.59	0.95	0.67	0.94	0.63	0.95
DevT2	0.49	0.96	0.41	0.95	0.45	0.96
DevK1	0.91	0.99	0.90	1.00	0.90	0.99
DevK2	0.95	0.99	0.87	0.99	0.91	0.99
PerFlow(Avg)	0.66	0.97	0.72	0.97	0.69	0.97

Table 18: Device Type - Known Analysis on Wireless Campus Network (Passive)

	Ping_C1		Ping_C2		PerDevice(Avg)	
	γ	α	γ	α	γ	α
AsusNetbook	0.97	0.91	0.99	0.98	0.98	0.94
Camera	0.53	0.93	0.88	0.95	0.71	0.94
Gaming	0.74	0.96	1.00	0.99	0.87	0.97
GooglePhone	0.81	0.96	0.75	0.95	0.78	0.96
Laptop	0.58	0.94	0.51	0.88	0.55	0.91
Printer	1.00	1.00	1.00	1.00	1.00	1.00
TV	0.12	0.86	0.26	0.90	0.19	0.88
Tablet	0.79	0.90	0.82	0.94	0.81	0.92
eReader	0.88	0.98	0.67	0.94	0.78	0.96
PerFlow(Avg)	0.71	0.94	0.77	0.95	0.74	0.94

A.2.2 Unknown Analysis

A.2.2.1 Passive Fingerprinting

Table 19: Device ID - Unknown Analysis on Wireless Campus Network (Passive)

	Ping_C1		Ping_C2		Tcp_C1		Udp_C1		UdpC3		Udp_C4		Scp_C1		Skype_C1		PerDevice(Avg)	
	γ	α	γ	α	γ	α	γ	α	γ	α	γ	α	γ	α	γ	α	γ	α
DevAS1	0.48	0.95	0.70	0.95	0.83	0.96	0.81	0.98	0.53	0.97	0.74	0.97	0.22	0.94	-NA-	-NA-	0.62	0.96
DevAS2	0.83	0.97	0.57	0.95	0.41	0.93	0.70	0.96	0.60	0.95	0.91	0.98	0.38	0.92	-NA-	-NA-	0.63	0.95
DevAS3	0.64	0.93	0.75	0.97	0.64	0.95	0.79	0.97	0.93	0.99	0.86	0.99	0.77	0.94	-NA-	-NA-	0.77	0.96
DevAS4	0.61	0.92	0.33	0.92	0.53	0.97	0.73	0.95	0.91	0.92	0.67	0.97	0.13	0.93	-NA-	-NA-	0.56	0.94
DevAS5	0.13	0.92	0.78	0.96	0.27	0.92	0.33	0.92	0.03	0.92	0.72	0.96	0.76	0.92	-NA-	-NA-	0.43	0.93
DevG1	0.78	0.98	1.00	1.00	0.52	0.93	0.94	0.99	0.86	0.99	0.84	0.98	0.79	0.99	-NA-	-NA-	0.82	0.98
DevG2	1.00	1.00	0.97	1.00	0.36	0.93	0.99	1.00	0.25	0.93	0.75	0.96	0.94	1.00	-NA-	-NA-	0.75	0.97
DevL1	0.60	0.95	0.16	0.94	0.83	0.98	0.72	0.96	0.84	0.99	0.75	0.98	0.56	0.95	0.74	0.94	0.65	0.96
DevL2	0.43	0.94	0.77	0.95	0.48	0.94	0.67	0.94	1.00	0.99	0.73	0.96	0.64	0.95	0.88	0.98	0.70	0.96
DevT1	0.68	0.91	0.87	0.97	0.29	0.91	0.66	0.93	0.21	0.93	0.29	0.92	0.53	0.94	0.66	0.85	0.52	0.92
DevT2	0.04	0.90	0.61	0.96	0.43	0.92	0.21	0.92	0.53	0.90	0.39	0.91	0.42	0.94	0.66	0.85	0.41	0.91
DevK1	1.00	1.00	0.94	1.00	0.84	0.99	-NA-	-NA-	0.22	0.94	-NA-	-NA-	0.85	0.99	-NA-	-NA-	0.77	0.98
DevK2	0.93	0.99	0.74	0.98	0.84	0.99	-NA-	-NA-	0.84	0.99	-NA-	-NA-	0.93	0.99	-NA-	-NA-	0.85	0.99
Unknown	0.48	0.84	0.59	0.85	0.55	0.80	0.67	0.85	0.66	0.81	0.72	0.82	0.63	0.83	0.59	0.77	0.61	0.82
PerFlow(Avg)	0.62	0.94	0.70	0.96	0.56	0.94	0.69	0.95	0.60	0.94	0.70	0.95	0.61	0.94	0.70	0.88	0.64	0.94

Table 20: Device Type - Unknown Analysis on Wireless Campus Network (Passive)

	Ping_C1		Ping_C2		Tcp_C1		Udp_C1		UdpC3		Udp_C4		Scp_C1		Skype_C1		PerDevice(Avg)	
	γ	α	γ	α	γ	α	γ	α	γ	α	γ	α	γ	α	γ	α	γ	α
AsusNetbook	0.97	0.99	0.93	0.99	0.90	0.98	0.87	0.98	0.80	0.95	0.90	0.98	0.69	0.95	-NA-	-NA-	0.87	0.97
GooglePhone	0.06	0.84	0.08	0.82	0.61	0.84	0.46	0.89	0.38	0.83	0.50	0.86	0.02	0.82	-NA-	-NA-	0.30	0.84
Laptop	0.92	0.95	0.94	0.99	0.45	0.79	0.77	0.91	0.42	0.86	0.03	0.73	0.93	0.99	0.84	0.96	0.66	0.90
Tablet	0.80	0.92	0.83	0.93	0.50	0.88	0.79	0.81	0.50	0.83	0.74	0.73	0.88	0.97	0.88	0.97	0.74	0.88
eReader	0.35	0.78	0.33	0.75	0.53	0.91	-NA-	-NA-	0.59	0.93	-NA-	-NA-	0.77	0.91	-NA-	-NA-	0.51	0.85
Unknown	0.72	0.79	0.53	0.74	0.55	0.78	0.67	0.82	0.61	0.71	0.40	0.72	0.81	0.73	1.00	0.93	0.66	0.78
PerFlow(Avg)	0.64	0.88	0.61	0.87	0.59	0.86	0.71	0.88	0.55	0.85	0.52	0.80	0.68	0.90	0.91	0.95	0.63	0.87

A.2.2.2 Active Analysis

Table 21: Device ID - Unknown Analysis on Wireless Campus Network (Active)

	Ping_C1		**Ping_C2**		**PerDevice(Avg)**	
	γ	α	γ	α	γ	α
DevAS1	0.43	0.97	0.64	0.97	0.54	0.97
DevAS2	0.66	0.95	0.81	0.99	0.73	0.97
DevAS3	0.38	0.95	0.55	0.95	0.46	0.95
DevAS4	0.48	0.95	0.38	0.95	0.43	0.95
DevAS5	0.40	0.96	0.37	0.96	0.38	0.96
DevC1	0.23	0.96	0.17	0.96	0.20	0.96
DevC2	0.47	0.97	0.70	0.97	0.59	0.97
DevP1	0.90	0.98	0.78	0.97	0.84	0.97
DevP2	0.01	0.94	0.15	0.96	0.08	0.95
DevG1	0.85	0.99	0.71	0.99	0.78	0.99
DevG2	0.54	0.97	0.72	0.98	0.63	0.98
DevL1	0.34	0.95	0.54	0.96	0.44	0.96
DevL2	0.77	0.98	0.60	0.98	0.69	0.98
DevH1	0.63	0.95	0.79	0.99	0.71	0.97
DevH2	0.05	0.96	0.74	0.98	0.39	0.97
DevA1	0.72	0.96	0.30	0.97	0.51	0.96
DevA2	0.42	0.97	0.98	1.00	0.70	0.98
DevT1	0.53	0.95	0.17	0.92	0.35	0.93
DevT2	0.28	0.94	0.42	0.95	0.35	0.94
DevK1	0.72	0.98	0.92	1.00	0.82	0.99
DevK2	0.93	0.99	0.78	0.99	0.85	0.99
Unknown	0.45	0.81	0.57	0.81	0.51	0.81
PerFlow(Avg)	0.51	0.96	0.58	0.96	0.55	0.96

Table 22: Device Type - Unknown Analysis on Wireless Campus Network (Active)

	Ping_C1		**Ping_C2**		**PerDevice(Avg)**	
	γ	α	γ	α	γ	α
AsusNetbook	0.83	0.96	0.85	0.98	0.84	0.97
Camera	0.41	0.93	0.71	0.96	0.56	0.94
Gaming	0.83	0.98	0.95	0.99	0.89	0.99
GooglePhone	0.71	0.97	0.72	0.96	0.72	0.97
Laptop	0.54	0.93	0.57	0.89	0.56	0.91
Printer	0.97	1.00	0.85	0.98	0.91	0.99
TV	0.17	0.90	0.17	0.90	0.17	0.90
Tablet	0.98	0.96	0.85	0.97	0.91	0.96
eReader	0.81	0.98	0.69	0.96	0.75	0.97
Unknown	0.78	0.81	0.77	0.82	0.78	0.81
PerFlow(Avg)	0.70	0.94	0.71	0.94	0.71	0.94

A.3 Results from Wired Campus Network Experiments

A.3.1 Known Analysis

A.3.1.1 Passive Fingerprinting

Table 23: Device ID - Known Analysis on Wired Campus Network (Passive)

	Ping_C1		**Ping_C2**		**Tcp_C1**		**Udp_C1**		**UdpC3**		**Udp_C4**		**PerDevice(Avg)**	
	γ	α	γ	α	γ	α	γ	α	γ	α	γ	α	γ	α
DevAS1	0.70	0.95	0.56	0.89	0.51	0.88	0.33	0.87	0.89	0.96	0.78	0.94	0.63	0.92
DevAS2	0.87	0.97	0.19	0.86	0.42	0.87	0.21	0.79	0.05	0.89	0.61	0.89	0.39	0.88
DevAS3	1.00	1.00	1.00	0.99	0.43	0.86	0.16	0.89	0.50	0.88	0.48	0.82	0.59	0.91
DevAS4	0.76	0.92	0.51	0.92	0.44	0.87	0.11	0.85	0.67	0.84	0.09	0.85	0.43	0.88
DevAS5	0.54	0.90	0.93	0.94	0.24	0.85	0.33	0.74	0.27	0.85	0.31	0.90	0.43	0.86
DevD1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
DevD2	0.99	0.99	0.92	0.99	1.00	1.00	0.83	0.93	0.20	0.85	0.59	0.91	0.75	0.95
DevL1	0.93	0.99	0.99	1.00	0.87	0.96	0.95	0.99	0.38	0.93	0.83	0.96	0.82	0.97
DevL2	0.94	0.99	0.96	0.98	0.75	0.96	0.62	0.94	0.95	0.89	0.65	0.92	0.81	0.95
PerFlow(Avg)	0.86	0.97	0.78	0.95	0.63	0.92	0.50	0.89	0.55	0.90	0.59	0.91	0.65	0.92

Table 24: Device Type - Known Analysis on Wired Campus Network (Passive)

	Ping_C1		**Ping_C2**		**Tcp_C1**		**Udp_C1**		**UdpC3**		**Udp_C4**		**PerDevice(Avg)**	
	γ	α	γ	α	γ	α	γ	α	γ	α	γ	α	γ	α
AsusNetbook	0.99	1.00	1.00	0.95	1.00	0.87	1.00	0.89	1.00	0.79	1.00	0.80	1.00	0.88
Dell	0.39	0.76	0.27	0.72	0.60	0.87	0.29	0.72	0.01	0.50	0.00	0.56	0.26	0.69
Laptop	0.87	0.76	0.89	0.76	1.00	1.00	0.86	0.83	0.45	0.69	0.69	0.76	0.80	0.80
PerFlow(Avg)	0.75	0.84	0.72	0.81	0.87	0.91	0.72	0.81	0.49	0.66	0.56	0.71	0.68	0.79

A.3.1.2 Active Fingerprinting

Table 25: Device ID - Known Analysis on Wired Campus Network (Active)

	Ping_C1		**Ping_C2**		**PerDevice(Avg)**	
	γ	α	γ	α	γ	α
DevAS1	0.53	0.93	0.38	0.93	0.46	0.93
DevAS2	0.42	0.94	0.34	0.91	0.38	0.92
DevAS3	0.49	0.93	0.94	0.98	0.72	0.95
DevAS4	0.68	0.95	0.27	0.92	0.47	0.93
DevAS5	0.22	0.94	0.20	0.93	0.21	0.93
DevC1	0.97	0.96	0.41	0.92	0.69	0.94
DevC2	0.34	0.96	0.16	0.91	0.25	0.94
DevD1	1.00	1.00	1.00	1.00	1.00	1.00
DevD2	1.00	1.00	0.95	0.99	0.98	1.00
DevP1	0.34	0.96	0.84	0.99	0.59	0.97
DevP2	0.99	0.96	0.99	0.99	0.99	0.97
DevL1	0.95	1.00	1.00	0.99	0.98	0.99
DevL2	1.00	1.00	0.71	0.98	0.86	0.99
DevH1	0.98	0.94	0.81	0.97	0.89	0.95
DevH2	0.01	0.94	0.61	0.96	0.31	0.95
DevA1	0.34	0.96	0.83	0.99	0.58	0.97
DevA2	0.94	0.96	0.92	0.99	0.93	0.97
PerFlow(Avg)	0.66	0.96	0.67	0.96	0.66	0.96

Table 26: Device Type - Known Analysis on Wired Campus Network (Active)

	Ping_C1		**Ping_C2**		**PerDevice(Avg)**	
	γ	α	γ	α	γ	α
AsusNetbook	1.00	1.00	1.00	0.92	1.00	0.96
Camera	0.99	0.94	1.00	0.96	1.00	0.95
Dell	0.13	0.87	0.03	0.86	0.08	0.87
Gaming	1.00	1.00	1.00	1.00	1.00	1.00
Laptop	1.00	0.99	0.70	0.96	0.85	0.97
Printer	0.99	0.94	1.00	0.95	1.00	0.94
TV	1.00	1.00	1.00	1.00	1.00	1.00
PerFlow(Avg)	0.87	0.96	0.82	0.95	0.85	0.96

A.3.2 Unknown Analysis

A.3.2.1 Passive Fingerprinting

Table 27: Device ID - Unknown Analysis on Wired Campus Network (Passive)

	Ping_C1		**Ping_C2**		**Tcp_C1**		**Udp_C1**		**UdpC3**		**Udp_C4**		**PerDevice(Avg)**	
	γ	α	γ	α	γ	α	γ	α	γ	α	γ	α	γ	α
DevAS1	0.37	0.92	0.55	0.93	0.50	0.90	0.26	0.89	0.52	0.95	0.80	0.96	0.50	0.92
DevAS2	0.63	0.94	0.18	0.87	0.34	0.88	0.17	0.80	0.00	0.90	0.78	0.94	0.35	0.89
DevAS3	0.91	0.99	0.90	0.99	0.30	0.88	0.03	0.90	0.80	0.93	0.73	0.87	0.61	0.93
DevAS4	0.61	0.92	0.25	0.90	0.41	0.88	0.06	0.88	0.60	0.86	0.07	0.88	0.33	0.89
DevAS5	0.53	0.89	0.80	0.94	0.28	0.86	0.24	0.77	0.26	0.87	0.25	0.91	0.39	0.87
DevD1	0.90	0.99	0.89	0.99	0.91	0.99	0.92	0.99	0.77	0.98	0.92	0.99	0.88	0.99
DevD2	0.91	0.99	0.79	0.98	0.89	0.99	0.53	0.93	0.04	0.85	0.56	0.93	0.62	0.94
DevL1	0.86	0.98	0.79	0.98	0.74	0.95	0.86	0.99	0.25	0.92	0.91	0.97	0.73	0.97
DevL2	0.77	0.98	0.90	0.99	0.58	0.95	0.65	0.95	0.89	0.90	0.66	0.95	0.74	0.95
Unknown	0.74	0.85	0.77	0.81	0.50	0.81	0.49	0.75	0.47	0.75	0.54	0.84	0.58	0.80
PerFlow(Avg)	0.72	0.94	0.68	0.94	0.55	0.91	0.42	0.88	0.46	0.89	0.62	0.92	0.58	0.92

Table 28: Device Type - Unknown Analysis on Wired Campus Network (Passive)

	Ping_C1		**Ping_C2**		**Tcp_C1**		**Udp_C1**		**UdpC3**		**Udp_C4**		**PerDevice(Avg)**	
	γ	α	γ	α	γ	α	γ	α	γ	α	γ	α	γ	α
AsusNetbook	0.76	0.93	0.77	0.95	0.90	0.98	0.89	0.97	0.85	0.87	0.86	0.96	0.84	0.94
Dell	0.27	0.71	0.30	0.78	0.80	0.96	0.52	0.84	0.22	0.68	0.40	0.77	0.42	0.79
Laptop	0.80	0.79	0.85	0.85	0.90	0.98	0.75	0.88	0.46	0.83	0.82	0.88	0.76	0.87
Unknown	0.37	0.64	0.68	0.73	1.00	0.91	0.80	0.79	0.49	0.63	0.65	0.75	0.66	0.74
PerFlow(Avg)	0.55	0.76	0.65	0.83	0.90	0.96	0.74	0.87	0.51	0.75	0.68	0.84	0.67	0.84

A.3.2.2 Active Fingerprinting

Table 29: Device ID - Unknown Analysis on Wired Campus Network (Active)

	Ping_C1		**Ping_C2**		**PerDevice(Avg)**	
	γ	α	γ	α	γ	α
DevAS1	0.55	0.94	0.24	0.93	0.39	0.93
DevAS2	0.24	0.94	0.22	0.91	0.23	0.93
DevAS3	0.29	0.93	0.79	0.97	0.54	0.95
DevAS4	0.80	0.98	0.18	0.93	0.49	0.95
DevAS5	0.58	0.94	0.32	0.93	0.45	0.94
DevC1	0.75	0.96	0.39	0.92	0.57	0.94
DevC2	0.03	0.95	0.22	0.93	0.13	0.94
DevD1	0.00	0.94	0.98	1.00	0.49	0.97
DevD2	0.87	0.99	0.85	0.99	0.86	0.99
DevP1	0.25	0.96	0.87	0.99	0.56	0.98
DevP2	0.92	0.96	0.89	0.99	0.91	0.97
DevL1	0.48	0.97	0.91	0.99	0.69	0.98
DevL2	0.53	0.92	0.69	0.97	0.61	0.94
DevH1	0.76	0.96	0.65	0.97	0.71	0.97
DevH2	0.08	0.95	0.70	0.97	0.39	0.96
DevA1	0.30	0.95	0.69	0.98	0.50	0.97
DevA2	0.85	0.96	0.85	0.98	0.85	0.97
Unknown	0.46	0.77	0.55	0.86	0.51	0.82
PerFlow(Avg)	0.49	0.94	0.61	0.96	0.55	0.95

Table 30: Device Type - Unknown Analysis on Wired Campus Network (Active)

	Ping_C1		**Ping_C2**		**PerDevice(Avg)**	
	γ	α	γ	α	γ	α
AsusNetbook	0.90	0.99	0.83	0.94	0.86	0.96
Camera	0.74	0.97	0.94	0.99	0.84	0.98
Dell	0.00	0.86	0.10	0.89	0.05	0.87
Gaming	1.00	1.00	0.87	0.98	0.93	0.99
Laptop	0.86	0.95	0.64	0.95	0.75	0.95
Printer	0.76	0.92	0.92	0.99	0.84	0.95
TV	0.92	0.99	0.89	0.99	0.90	0.99
Unknown	0.80	0.82	0.93	0.80	0.86	0.81
PerFlow(Avg)	0.75	0.94	0.77	0.94	0.76	0.94

REFERENCES

[1] K. Gao, C. L. Corbett, and R. A. Beyah, "A passive approach to wireless device fingerprinting," 2010, johns Hopkins Applied Physics Lab, White Paper.

[2] J. Predd, S. Pfleeger, J. Hunker, and C. Bulford, "Insiders behaving badly," *Security Privacy, IEEE*, vol. 6, no. 4, pp. 66 –70, july-aug. 2008.

[3] R. Rantala, "Cybercrime against businesses," 2005, `http://bjs.ojp.usdoj.gov/content/pub/pdf/cb05.pdf`.

[4] "Mcafee network access control 3.1 product and installation guide for use with epolicy orchestrator 4.0," 2009, `McAfee, Whitepaper`.

[5] "Network security scanner," `http://www.nmap.org/`.

[6] "IEEE oui list," `http://standards.ieee.org/develop/regauth/oui/oui.txt`.

[7] "Xprobe: Active os fingerprinting tool," `http://xprobe.sourceforge.net/`.

[8] F. Yarochkin, O. Arkin, M. Kydyraliev, S.-Y. Dai, Y. Huang, and S.-Y. Kuo, "Xprobe2++: Low volume remote network information gathering tool," in *Dependable Systems Networks, 2009. DSN '09. IEEE/IFIP International Conference on*, 29 2009-July 2, pp. 205–210.

[9] "TCP/IP protocol based passive fingerprinting," `http://lcamtuf.coredump.cx/p0f.shtml`.

[10] "SinFP: A new approach to os fingerprinting," `http://www.gomor.org/bin/view/Sinfp/WebHome`.

[11] J. Medeiros, A. da Cunha, A. Brito, and P. Motta Pires, "Automating security tests for industrial automation devices using neural networks," in *Emerging Technologies and Factory Automation, 2007. ETFA. IEEE Conference on*, Sept., pp. 772–775.

[12] D. W. Richardson, S. D. Gribble, and T. Kohno, "The limits of automatic os fingerprint generation," in *Proceedings of the 3rd ACM workshop on Artificial intelligence and security*, 2010, pp. 24–34.

[13] X. Zhang and L. Zheng, "Delude remote operating system (os) scan by honeyd," in *Computer Science and Engineering, 2009. WCSE '09. Second International Workshop on*, vol. 2, Oct., pp. 503–506.

[14] S. Kalia and M. Singh, "Masking approach to secure systems from operating system fingerprinting," in *IEEE Region 10 TENCON*, 2005, pp. 1 – 6.

[15] T. B. A. Kohno and K. C. Claffy, "Remote physical device fingerprinting," in *Proc. of the 2005 IEEE Symposium on Security and Privacy*, Washington, DC, USA, pp. 211–225.

[16] S. Jana and S. K. Kasera, "On fast and accurate detection of unauthorized wireless access points using clock skews," in *MobiCom '08: Proc. of the 14th ACM International Conf. on Mobile computing and networking*, pp. 104–115.

[17] V. Brik, S. Banerjee, M. Gruteser, and S. Oh, "Wireless device identification with radiometric signatures," in *Proc. of the 14th ACM International Conf. on Mobile Computing and Networking (MobiCom)*, 2008.

[18] J. Hall, M. Barbeau, and E. Kranakis, "Rogue devices in bluetooth networks using radio frequency fingerprinting," in *IASTED International Conf. on Communications and Computer Networks (CCN)*, 2006.

[19] S. Bratus, C. Cornelius, D. Kotz, and D. Peebles, "Active behavioral fingerprinting of wireless devices," in *ACM WiSec '08: Proc. of the first ACM conference on Wireless network security*, pp. 56–61.

[20] K. Gao, C. Corbett, and R. Beyah, "A passive approach to wireless device fingerprinting," in *Dependable Systems and Networks (DSN), 2010 IEEE/IFIP International Conference on*, 28 2010-july 1 2010, pp. 383 –392.

[21] L. Letaw, J. Pletcher, and K. Butler, "Host identification via usb fingerprinting," *Systematic Approaches to Digital Forensic Engineering (SADFE)*, 2011.

[22] J. Francois, H. Abdelnurt, R. State, and O. Festort, "Ptf: Passive temporal fingerprinting," in *International Symposium on Integrated Network Management (IM)*, 2011.

[23] J. Francois, R. State, H. Abdelnurt, and O. Festort, "Machine learning techniques for passive network inventory," in *IEEE Transactions on Network and Service Management*, vol. 7, 2010.

[24] J. Francois, T. Engel, R. State, and O. Festort, "Enforcing security with behavioral fingerprinting," in *International Conf. on Network and Service Management (CNSM)*, 2011.

[25] J. Francois, R. State, T. Engel, and O. Festor, "Enforcing security with behavioral fingerprinting," in *Network and Service Management (CNSM), 2011 7th International Conference on*, Oct., pp. 1–9.

[26] "IANA port number list," `http://www.iana.org/assignments/port-20 numbers`.

[27] T. Karagiannis, K. Papagiannaki, and M. Faloutsos, "Blinc: multilevel traffic classification in the dark," in *Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, 2005, pp. 229–240.

[28] M. Kim, Y. J. Won, and J. W. Hong, "Application-level traffic monitoring and an analysis on IP networks," *ETRI Journal*, vol. 27, 2005.

[29] A. Moore and D. Zuev, "Internet traffic classification using bayesian analysis techniques," in *Proc. of the 2005 ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, pp. 50 – 60.

[30] M. Roughan, S. Sen, O. Spatscheck, and N. Duffield, "Class-of-service mapping for qos: a statistical signature-based approach to ip traffic classification," in *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, 2004, pp. 135–148.

[31] “Tcpdump: A packet analyzer tool,” 2009, `http://www.tcpdump.org/`.

[32] “Airmon-ng: Wireless interface setup tool,” `http://www.aircrack-ng.org`.

[33] H. Kim, V. S. Pai, and S. Rixner, “Exploiting task-level concurrency in a programmable network interface,” in *Proceedings of the ninth ACM SIGPLAN symposium on Principles and practice of parallel programming*, 2003, pp. 61–72.

[34] S. Sathyanarayana, “Characterizing the effects of device components on network traffic,” *Master’s Thesis - Georgia Institute of Technology*, 2013.

[35] G. Kakavelakis, R. Beverly, and J. Young, “Auto-learning of smtp tcp transport-layer features for spam and abusive message detection,” in *Proceedings of the 25th international conference on Large Installation System Administration*, 2011, pp. 18–18.

[36] “Sharktools,” http://www.mit.edu/ armenb/sharktools/.

VITA

Sakthi Vignesh Radhakrishnan from Chennai, India, obtained his Bachelors degree in Electronics and Communication Engineering from SSN College of Engineering (Anna University) in 2011. During his Master's at Georgia Institute of Technology, he was a member of the Communication Assurance and Performance (CAP) group which focuses on problems related to network security and performance. In May 2013, Sakthi graduated with a Master's degree in Electrical and Computer Engineering from Georgia Institute of Technology and joined Qualcomm Inc as a firmware developer for corporate routers. His research interests include network security and wireless networking.

Publications :

[1] S.V. Radhakrishnan, and Subramanian S, "An analytical approach to s-box generation," *Journal on Computer and Electrical Engineering*, (2012)

[2] S. Uluagac, S.V. Radhakrishnan, C. Corbett, A. Beca, and R. Beyah, "A Passive Technique for Fingerprinting Wireless Devices with Wireside Observations," in *IEEE International Conference on Communications and Network Security (CNS)*, 2013 - in submission

[3] S.V. Radhakrishnan, S. Uluagac, and R. Beyah, "Realizing an 802.11-based Covert Timing Channel Using Off-The-Shelf Wireless Cards," in *IEEE International Symposium on a World of Wireless Mobile and Multimedia Networks (WoWMoM)*, 2013 - in submission

[4] S.V. Radhakrishnan, S. Uluagac, and R. Beyah, "Networked System Fingerprinting," *IEEE Security and Privacy Magazine* - in preparation