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

Sakthi Vignesh Radhakrishnan, *Student Member, IEEE*, A. Selcuk Uluagac, *Senior Member, IEEE*, and Raheem Beyah, *Senior Member, IEEE*

Abstract—In this paper, we introduce GTID, a technique that can actively and passively fingerprint wireless devices and their types using wire-side observations in a local network. GTID exploits information that is leaked as a result of heterogeneity in devices, which is a function of different device hardware compositions and variations in devices' clock skew. We apply statistical techniques on network traffic to create unique, reproducible device and device type signatures, and use artificial neural networks (ANNs) for classification. We demonstrate the efficacy of our technique on both an isolated testbed and a live campus network (during peak hours) using a corpus of 37 devices representing a wide range of device classes (e.g., iPads, iPhones, Google Phones, etc.) and traffic types (e.g., Skype, SCP, ICMP, etc.). Our experiments provided more than 300 GB of traffic captures which we used for ANN training and performance evaluation. In order for any fingerprinting technique to be practical, it must be able to detect previously unseen devices (i.e., devices for which no stored signature is available) and must be able to withstand various attacks. GTID is a fingerprinting technique to detect previously unseen devices and to illustrate its resilience under various attacker models. We measure the performance of GTID by considering accuracy, recall, and processing time and also illustrate how it can be used to complement existing security mechanisms (e.g., authentication systems) and to detect counterfeit devices.

Index Terms—GTID, device fingerprinting, wireless device fingerprinting, device type fingerprinting

1 Introduction

Identifying devices connected to a network (i.e., *device fingerprinting*) has become of critical importance to ensure security services to the network. In the same vein, there has also been a need to understand the type of a device that is connected to a network (i.e., *device type fingerprinting*). Device fingerprinting seeks to uniquely identify devices on a network without considering existing easily forgeable identifiers (e.g., Internet Protocol (IP) and Medium Access Control (MAC) addresses). On the other hand, device type fingerprinting can be used to determine if a device belongs to a particular cohort.

In this paper, we present *GTID*[1]*: A Technique for Physical Device and Device Type Fingerprinting*. Our technique uses information leaked by the physical implementation of a device through its network traffic to identify a device and a device's type. This is accomplished by exploiting the heterogeneity of devices which is a function of the different hardware compositions (e.g., processor, DMA controller, memory) of the devices as well as the clock skews. While the former enables device type identification, the latter enables device identification. We use statistical techniques to capture time-variant behavior of network traffic and to create unique, reproducible device and device type signatures. For this, GTID passively collects traffic on a wired segment between the access point (AP) and the final destination in a local network and uses Artificial Neural Network (ANN) based algorithm to identify devices and their types. Unlike current wireless intrusion detection systems (WIDSs) or wireless device fingerprinting techniques that monitor the wireless spectrum, our technique allows for wireless device and device type fingerprinting from the wired side of the network (e.g., at a backbone switch). Thus, GTID can alleviate the need for costly spectrum analyzers and the need to be within wireless range of the device to be fingerprinted. In addition, GTID can complement existing security solutions such as authentication, network management, and access control systems.

In general, although it is possible to combine existing device and device type fingerprinting techniques, current fingerprinting techniques either identify unique devices [1], [2], [3], [4], [5], [6] or identify device types [7], [8], [9], [10], [11], [12]. However, a technique that does both can prove very useful. For example, the device ID capability (device fingerprinting) provided by GTID can be used to augment existing Network Access Control (NAC) systems that seek to control access to a network. GTID can be used as a standalone fingerprinting system; however when used in conjunction with an exiting NAC, two limitations of current NAC systems can be obviated. Given that NAC client software passes the user's credentials to a backend server (e.g., RADIUS) for authentication, the user is ultimately authorized, not the device. Therefore, a user can simply transfer his or her credentials to another device, which could be

1. The name of our technique is inspired by Georgia Tech ID cards.

• *A.S. Uluagac is with the Department of Electrical and Computer Engineering, Florida International University, Miami, FL. E-mail: suluagac@fiu.edu.*
• *S.V. Radhakrishnan and R.A. Beyah are with the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30318. E-mail: sakthi03@gatech.edu, rbeyah@ece.gatech.edu.*

Manuscript received 28 Sept. 2013; revised 30 May 2014; accepted 26 Sept. 2014. Date of publication 9 Nov. 2014; date of current version 16 Sept. 2015. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TDSC.2014.2369033

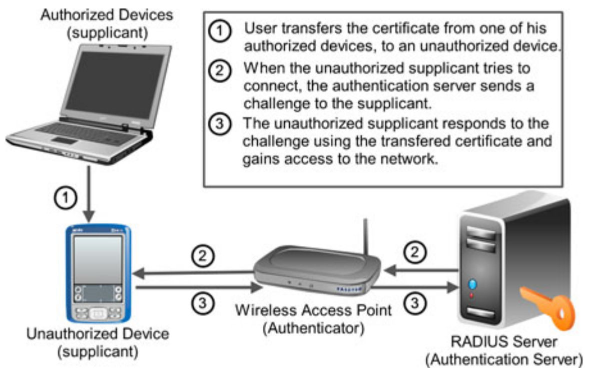

Authorized Devices (supplicant)
① User transfers the certificate from one of his authorized devices, to an unauthorized device.
② When the unauthorized supplicant tries to connect, the authentication server sends a challenge to the supplicant.
③ The unauthorized supplicant responds to the challenge using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

Fig. 1. Authentication of an unauthorized device in 802.1x.

unauthorized, and access the network in a legitimate fashion (Fig. 1). On the other hand, because GTID's device type fingerprinting capability detects differences in devices internal composition, GTID can be used as a non-destructive method for counterfeit device detection (Fig. 2a)[2] [14]. With GTID used in conjunction with a NAC, the device can be authorized in addition to the user's credentials (Fig. 2b). A second limitation of current NAC systems is that they support a limited range of devices (e.g., Windows machines and Macs), and deem many other devices (e.g., IP printers, IP thermostats, gaming systems) as unmanageable. Since GTID uses information leaked by patterns in the network traffic, it does not require a software client and can thus be used on devices that are traditionally deemed as unmanageable (Fig. 2c). Note that as we approach the Internet of Things (IoTs), most devices will be considered unmanageable and may not work with current NAC solutions. The complete realization of each of the aforementioned services requires a highly accurate device fingerprinting technique. While GTID is not perfect, this paper presents the initial results of a promising system that takes a unique approach to device and device type fingerprinting.

The contributions of GTID are as follows: It (1) can provide device *and* device type fingerprinting; (2) is resilient to various attacker types; (3) can detect previously seen and *unknown* devices; (4) provides *wired-side detection* of wireless devices and device types; (5) works for various protocols (e.g., TCP, UDP, ICMP) and operating systems; (6) does not require deep packet inspection (e.g., to obtain timestamps or protocol banners), thus it is scalable and privacy preserving; (7) works on IP-level encrypted streams; (8) helps to ensure that devices be authorized, and not just the users (i.e., independent of user credentials); (9) does not require third party client software and works for devices that do not (or will not) have NAC clients although it can complement such existing security mechanisms. In this paper, we demonstrate the effectiveness and the practicality of GTID on both an isolated testbed and a live campus network using a collection of 37 devices belonging to a diverse set of device classes including iPads, iPhones, Kindles, Google-Phones, Netbooks, etc. The performance of GTID is evaluated by considering accuracy, recall, and processing time with real traffic from various applications and protocols such as Skype, ICMP, SCP, Iperf. We also consider eight different attack models and show how GTID would react to each of these attacks.

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 supplementing the significant challenge of local network access control (and other local network activities, e.g., counterfeit detection). Although the cost of equipment and experimentation was significant, the authors recognize that the performance of the system using a cohort of 37 devices is not representative of a network with 1,000+ devices. However, given that most business are small (87 percent of US companies have 19 or fewer employees while only 0.34 percent of US companies have 500 or more employees) [15], most networks are small, which implies that GTID could be useful for a significant segment of the deployed networks.

The remainder of this paper proceeds as follows. We first discuss the related work in Section 3. In Section 2, we discuss the background and theory behind our technique. Threat model and assumptions are articulated in Section 4. Section 5 provides an overview of the technique. In Section 6, we evaluate GTID while considering various attacker models. We analyze the limitations of GTID in Section 7 and then discuss its real-time implementation and performance analysis in Section 8. We conclude the paper and discuss future work in Section 9.

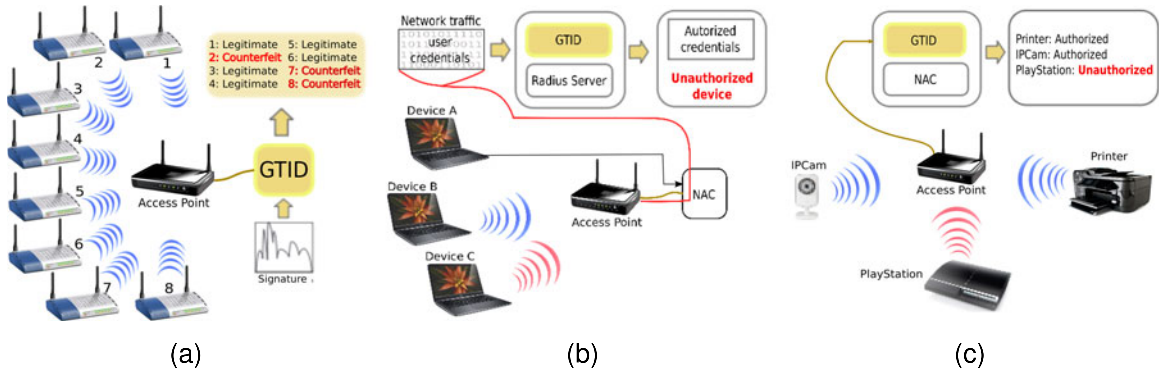


Fig. 2. How can device and type fingerprinting can complement other security security mechanisms? (a) Counterfeit device detection; (b) Device and user authentication with NAC; (c) Authentication of unmanageable devices with NAC.

2. Counterfeiting accounts for at least $7.5B in lost revenue for U.S. semiconductor companies [13].

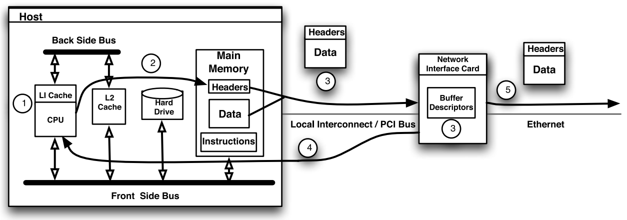


Fig. 3. Packet flow in hardware.

2 Background Discussion

Packet creation in a device is a complex process. It involves many internal parts of the device working together as shown in Fig. 3. Before a packet can be sent, the instruction set initiating the process is 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 the main memory, which contains the starting memory address and length of the packet to be sent. Multiple buffer descriptors are created if the packet is located in multiple discontiguous 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. 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 unit [16]. 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 DMA controller, the front side bus, the back side bus, the PCI bus, and the NIC. Therefore, the opportunities for diversity exist at both the device level, aiding device type identification, and at the component level, aiding device identification. At the device level, different vendors use different components with different capabilities and algorithms (e.g., Dell Latitude 2110 with Intel Atom N470 processor @ 1.83 GHz vs. Lenovo G570 with Intel Core i5-2430 processor @ 2.4 GHz) to create a device's internal architecture. Accordingly, the packet creation process varies across architectures aiding in device type identification.

In order to better understand its impact, we studied the variation in packet inter-arrival time (IAT) patterns captured for different CPU configurations and clock frequencies. The results of these experiments are presented in Figs. 4a and 4b. Fig. 4a shows the difference in the IAT patterns for two cache configurations (cache-config 1: data cache size 4KB, instruction cache uses Least Recently Used (LRU) replacement algorithm [17]; cache-config 2: data cache size 8KB, instruction cache uses Random replacement algorithm [17]), and Fig. 4b shows the result for three different clock frequencies. These experiments clearly demonstrate the impact of differences in hardware configuration on packet IATs. Moreover, at the component level, microscopic differences in the clock frequencies (clock skews) can contribute to 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 between two devices. Fig. 4c shows a plot of average IAT values, of four identical Asus netbooks generating 56 byte ping packets at 10ms intevals. The x-axis denotes the sample numbers, where each sample consists of 2.5 K ping packets. For instance, 20 samples will have 50 K 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 four 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.

3 Related Work

The existing work in this area can be broadly classified as device fingeperinting and device type fingerprinting.

The first category of fingerprinting is host or physical device fingerprinting. An important work in this area was presented by Kohno et al. in [1]. In [1], they introduce a method for fingerprinting physical devices by exploiting the implementation of the TCP protocol stack. 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. In contrast to the work in [1], our work is independent of protocol (i.e., it works for TCP, UDP, and

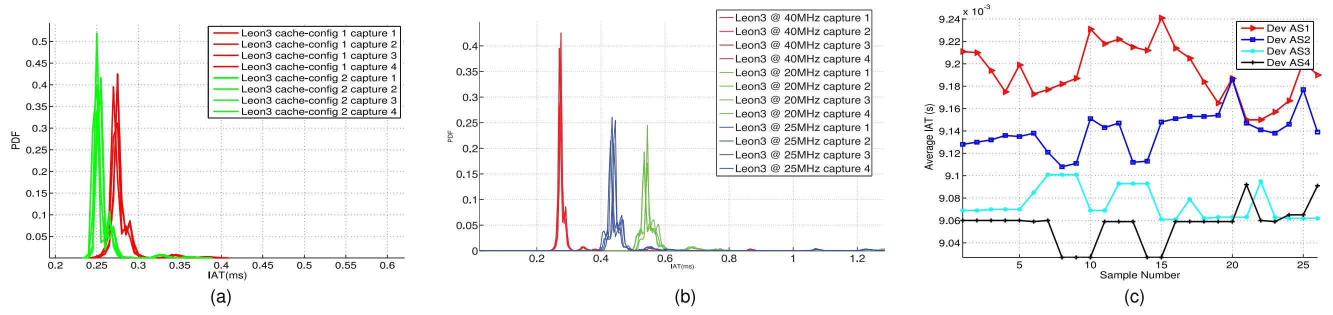


Fig. 4. (a) PDFs of IATs for different CPU configurations; (b) PDFs of IATs for different clock frequencies; (c) Effect of clock skew on IATs.

ICMP), does not require deep packet inspection (e.g., timestamps), thus it is more positioned for scalability and does not compromise privacy and works on IP level encrypted streams. The authors of [2] take a similar approach to that in [1] (i.e., using clock skew to uniquely identify nodes), however the goal of [2] is to uniquely fingerprint APs. Also, instead of getting the timestamp from TCP packets, they obtain the timestamp from 802.11 beacon frames. Another recent work in AP fingerprinting was done by the authors of [3]. The major improvement in [3] over [2] is that their technique is online and does not carry the fingerprint of the fingerprinting device. Since both the techniques make use of the 802.11 beacons, they can be used only for AP fingerprinting and cannot be used for general device fingerprinting. In [4], the authors evaluate the use of traffic parameters such as transmission rate, frame size, medium access time, transmission time, and inter arrival of packets to fingerprint 802.11 devices. However, these analysis are made with direct wireless side captures, which means an observer has to be in wireless range of the targeted device in order for this technique to work. Moreover, their proposed technique does not identify traffic from unseen sources to be coming from an unknown device. This is a major disadvantage compared to our technique because one cannot expect a system to possess signatures of all possible devices that it may encounter. 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 [5] and Bluetooth [6] emitters. The EM properties fingerprint the unique transmitter of a signal and differ from emitter to emitter. This technique requires expensive signal analyzer hardware to be within RF range of the target. In contrast, our approach only needs a network tap at a switch to capture traffic on a wired segment that could be a hop downstream.

Another body of work that is relevant to our work 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 [7], we introduce an active 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. In contrast to our current work, our previous work requires specially crafted data streams to trigger a signature. Further, the study was limited to AP types whereas our current scheme works with normal traffic across a wide range of device types, and can be used to fingerprint devices as well as device types. Another work for fingerprinting of wireless APs with similar issues is introduced in [8], where the authors fingerprint wireless AP types by actively probing them with various regular and malformed packets. The authors of [9] 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 of 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. Additionally, in their early work [10], their technique required knowledge of the entire syntax of the protocol. In [9], this requirement is relaxed as they only need a corpus containing SIP sessions. The authors of [9], [10] develop a real-time approach and discuss deploying their techniques in [11]. However, all of their proposed techniques are limited to a specific application layer protocol - SIP, whereas our scheme works for ICMP, UDP, and TCP protocols and for the applications that they transport (e.g., Skype). The authors in [12] 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. One limitation of this work is that it requires one to be in physical possession of the device.

In general, our work is fundamentally different from the previous studies in several ways. The existing studies (1) fingerprint only either device types or devices, not both; (2) do not detect unknown devices (devices which do not have a signature); and (3) do not consider attackers who seek to disrupt the classification process.

4 Threat Model and Assumptions

As shown in Fig. 5, in our setup, a device transmits data over the air, which then gets forwarded through the backbone switch. GTID passively collects timing traffic from captured packets on the wired segment to identify the devices that are transmitting.

Since our technique is based on timing analysis (Section 5), 8 different classes of attackers are considered. The first seven attackers are considered to be *novice attackers* who have some knowledge of the detection technique and are capable of controlling their device's network traffic. These attackers can (1) *introduce constant delays to packet stream*; (2) *inject random delays to packets*; (3) *vary the packet size*; (4) *change the data rate*; (5) *modify/change the operating system*; (6) *load the CPU with intensive applications to over shadow normal behavior*; (7) *tunnel packets through another protocol*. The eighth attacker is assumed to be highly skilled and knowledgable of the technique. Hence, this attacker could try to emulate an authorized device's traffic pattern in order to establish/maintain network

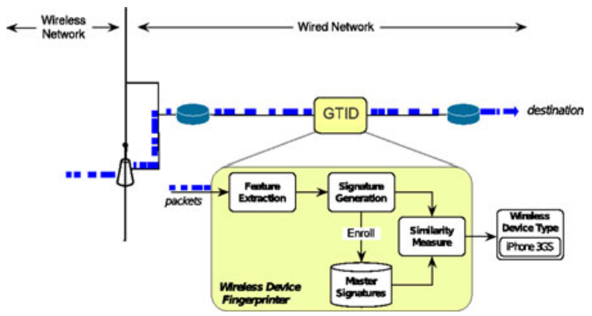


Fig. 5. Overview of GTID.

access. Analysis of these eight attacks and how GTID is resilient against these different threats are further discussed in Section 6.4.

5 Overview of GTID

In this section, we introduce the major components of GTID and discuss the Artificial Neural Network based algorithm used to identify devices and their types. We then articulate the metrics that we have used in evaluating the overall performance of the technique.

5.1 Components of the Technique

GTID has four major components: *feature extraction*, *signature generation*, *similarity measure*, and *enroll*.

Feature extraction. As traffic from devices are collected, the feature extraction process Measures traffic properties successively in time. The resulting feature vector with time values is 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. For our analysis, we use the packet inter-arrival time as our feature. IAT measures the delay (Δt) between successive packets and characterizes the traffic rate. The IAT feature vector is defined as

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

where Δ_{t_i} is the inter-arrival time between packet i and $i-1$.

Signature generation. The signature generation process uses statistical analysis to reveal patterns embedded in the traffic measurements. We adopted 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. We define frequency count as a vector that holds the number of IAT values falling within each of the N equally spaced time bins. 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 our experiments, we empirically determined $N = 300$ to be an ideal choice for all traffic types tested in this paper. We use this value of N to determine the binwidths for each traffic type.[3]

Enroll. Signatures generated from the previous component is used to train Artificial Neural Networks 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. Although in our work we treat the ANN as a black box, we specifically use ANNs that belong to a class called *feedforward networks*, where there is only a one-way connection from the input to the output layer [18]. ANNs belonging to this class require supervised training and are commonly used for prediction, *pattern recognition* and nonlinear function fitting. We configure our feedforward ANN to use *scaled conjugate gradient backpropagation* as the training function. We also note the use of *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.

3. The start and end points of the histogram are selected to fit the peak of certain traffic. This range is then split into 300 equal-sized bins.

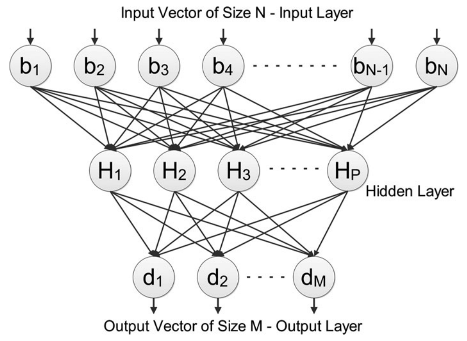

Input Vector of Size N - Input Layer
b1
b2
b3
b4
bN-1
bN
H1
H2
H3
HP
Hidden Layer
d1
d2
dM
Output Vector of Size M - Output Layer

Fig. 6. Illustration of a sample neural network.

Fig. 6 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. The input layer accepts 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 empirically determined to be 50. Two neural networks of this kind are used for each traffic type that we analyze. 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.

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.

5.2 Identification of Devices and Their Types

As explained earlier, GTID compares a device in question with previously collected master signatures and identifies the device in question and/or its type. We refer to the successful identification of an unknown device with one of the master devices as *Device Identification* and the identification of the unknown device's type with one of the master device types 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. In the first case, 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 case, 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. We note this result to be very meaningful because a detection system cannot hold all possible signatures of devices or types. There might be situations where the system will have to encounter signatures that it has never encountered in the past. In cases like these, GTID will rightly classify the signatures as unknown targets, unlike techniques that always pick the closest matching signature (Section 3). Finally, 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, the system extracts the neural networks ($\Theta_{ID} : DeviceID$, $\Theta_{Type} : DeviceType$) 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 Fig. 7. From the Kindle and eReader TP distributions in Fig. 7, 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

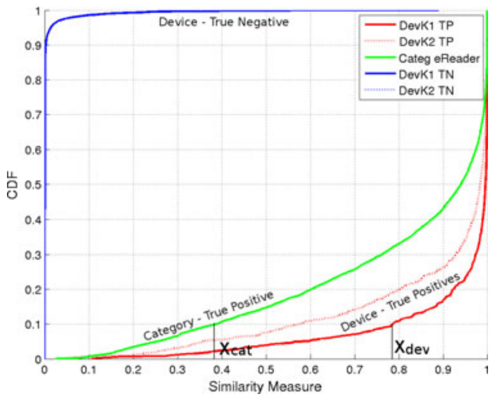


Fig. 7. CDF of closeness values for Kindle-Fire.

observed difference in the distribution patterns of device TP, the device type TP and TN from other devices (Fig. 7) 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 as discussed in Section 2. 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.

Algorithm 1. Device ID and Type Identification

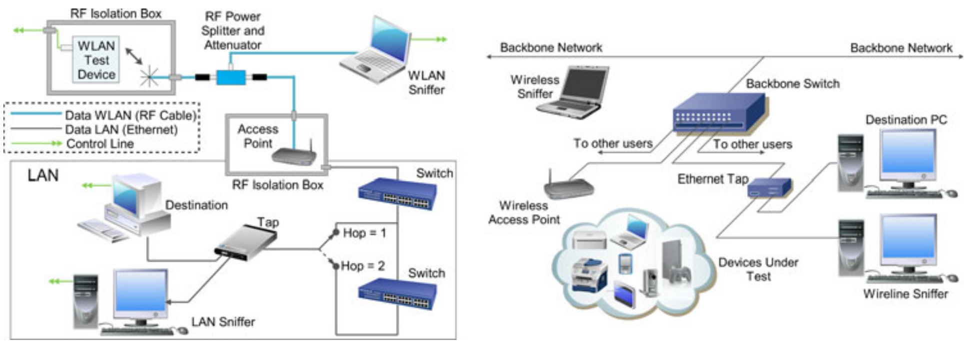
```
1:  $Identify-ID-Type()$
2:  begin
3:      $\Theta_{ID}, \Theta_{Type}, Dev_{list}, Type_{list} \leftarrow MastersDB(Traffic\ Type)$
4:      $\vec{\lambda}_{ID}, \vec{\lambda}_{Type} \leftarrow MaskingVectors()$
5:      $S \leftarrow IAT\_Sample()$
6:      $\Omega \leftarrow generate\_signature(S)$
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
```

5.3 Metrics for GTID's Effectiveness

We evaluate the performance of GTID using *accuracy* and *recall* as our metrics similar to [19]. 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. With accuracy, we measure the overall performance of our



(a) Isolated Network (b) Campus Network

Fig. 8. Setup of campus and isolated network testbeds.

system. Recall is the measure of identifying an actual device and it is statistically defined as

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

We 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.

6 Performance Evaluation

In this section, we evaluate the performance of GTID across four dimensions. First, we analyze our technique in an isolated network environment (Fig. 8a). Second, we measure the performance of GTID in a live campus network (Fig. 8b) during peak hours. Third, we evaluate the performance of a prototype version of GTID. Finally, we analyze the effectiveness of GTID under various attack scenarios in a live network.

In GTID, 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 shown in Fig. 8a, a control machine (not shown in the figure) was used to send commands to the different devices in the testbed. The device under test was placed in an isolation box to reduce RF leakage and interference. For the campus network testbed (Fig. 8b), the AP and LAN destination were connected to a campus backbone switch. This helped us evaluate our technique under MAC and physical layer interference from other wireless users in proximity (during peak hours). A total of more than 300 GB of traffic captures from 37 different devices [20], (14 in the isolated testbed, 23 in the campus network testbed) were tested and the details of which are listed in Tables 2 and 3.

Furthermore, two generic applications were used to generate traffic in our testbeds (Table 1). One was *Iperf*, which was used to generate both TCP and UDP traffic at controlled rates, and the other was *Ping*. In addition to these, we performed tests using other applications such as secure copy (*SCP*) and *Skype*. *TCP*, *SCP*, and *Skype* were allowed to flow at their natural rate, while *Ping* and *UDP* were controlled. In our experiments using Ping, we set the rate to *100 pings/second* and tested payload sizes of *64 Bytes* and *1,400 Bytes*. For *UDP* analysis we used two payload sizes, *64 Bytes* and *1,400 Bytes*, and sending rates of *1* and *8 Mbps*. Also, note that we classify all the above traffic types as either *Active* or *Passive*. Active traffic types are generated from the target in response to a trigger. For example, pinging a target device will result in ping responses (Active Traffic), which can then be fingerprinted (Active Fingerprinting). The passive traffic types are cases where the target system generates traffic without any trigger, e.g., a computer uploading data to a server. In these cases, the fingerprinting of such traffic is termed as passive fingerprinting. Although our experiments did not involve a human-generated traffic due to device use, the passive traffic is representative of experimental cases where human-generated traffic would be involved. Note that for each protocol/application in our tests, we only focused on one application/protocol without combining any protocols/applications. We captured 1 hour of traffic for each of these traffic types from 37 devices. This provided us more with more than 400 hours of traffic (about 300 GB of captures), which was used to

TABLE 1
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

TABLE 2
Isolated Network Device Specifications

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

TABLE 3
Campus Network Device Specifications

Device	Device ID	Model	Hardware Specification	Operating System	Kernel
Asus Netbook	AS1 AS2 AS3 AS4 AS5	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
Lenovo	L1 L2	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
Dell	D1 D2	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
ASUS Tablet	T1 T2	ASUS Transformer TF 101	1.0GHz NVIDIA Tegra 2 dual-core CPU 1GB RAM	Android 3.2.1	Kernel 2.6.36.3
Google Nexus One	G1 G2	Nexus One	1 GHz Qualcomm QSD8250 Processor 512MB RAM	Android 2.2	Kernel 2.6.32.9 Kernel 2.6.29
Kindle Fire	K1 K2	Kindle Fire	1 GHz Texas Instruments OMAP 4430 dual-core processor; 512MB RAM	Customized Android 2.3	Firmware 6.2.2 Firmware 6.2.1
Apple TV	A1 A2	ATV 1st Gen	Intel Pentium M processor 256MB DDR2 RAM @400MHz	OS Version 2.0 based on Mac OS X	-
HP Printer	H1 H2	HP Officejet 6500A Plus	-	RTOS	-
D-Link IP-Camera	C1 C2	D-Link DSC 932L	-	RTOS	-
PS3	P1 P2	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

evaluate GTID. The first half of each capture is used for training the ANN and the second half is used for performance analysis.

Note that before GTID can be applied on the traffic captures, it needs to be configured. There are two important values that are required to be set. The first one is the sample size, which denotes the number of IAT values that is used to generate a signature. Though it is better to have a larger sample size from an accuracy stand point, we note that it increases the processing time. Fig. 9a shows the variation in the recall of GTID and the average processing time per sample as the sample size is increased. It can be observed that the average time required for processing a sample increases almost linearly while the performance improvement follows a saturating curve. In our experiments, we chose the sample size to be 2.5K because it provided a recall of 74 percent[4] for a processing time of 120 ms, which is quite acceptable. Increasing the sample size beyond this does not provide any fruitful increase in

4. For known analysis.

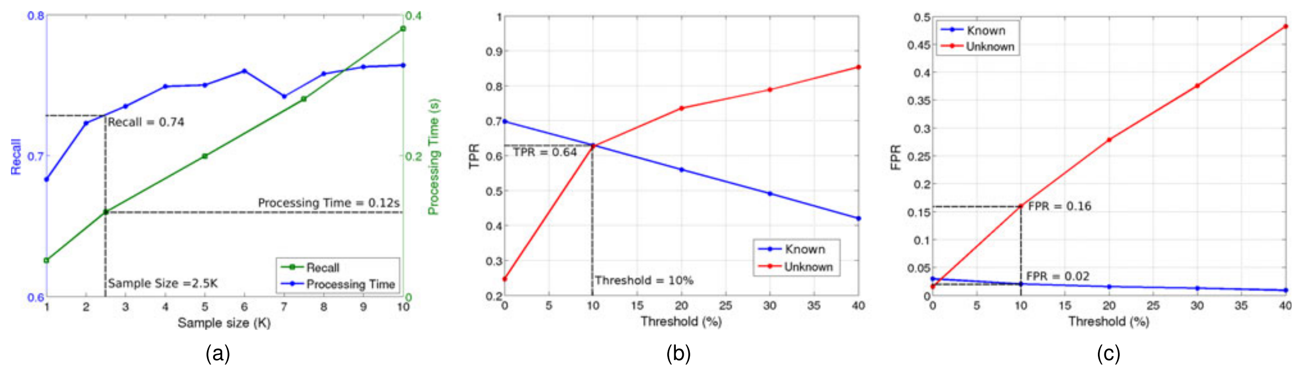


Fig. 9. Analysis of configurable parameters: (a) Sample size analysis (b) Variation in TPR vs. Unknown Threshold (c) Variation in FPR vs. Unknown Threshold.

performance. For instance, if the sample size is doubled to 5 K, the processing time increases by 66 percent while the value of recall increases by only 3 percent. The second parameter that needs to be set is the value for the 'Unknown' threshold. This value was empirically set to 10 percent which means the resulting value of closeness needs to be within the 90 percentile of previously seen closeness values for that particular device. The reason for choosing this value can be explained using Figs. 9b and 9c. Fig. 9b shows the variation in the true positive rate (TPR) of finding known and unknown devices as the threshold is increased and Fig. 9c shows the corresponding values for the false positive rate (FPR). From Fig. 9b, it can be seen that an equilibrium of guessing a known and an unknown device is reached for a threshold of 10 percent while maintaining the corresponding values of FPR at acceptable levels (Fig. 9c). It should be noted that the value of threshold can be moved around depending on the type of application. For example, if the network administrator is more concerned about finding unknown devices, he/she can move the threshold value higher to reduce the number of False Negatives.

As briefly mentioned in the previous section, 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* 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, we exposed GTID to both devices it has *previously seen* and devices that it has *not previously seen*. Note that for this, we took out the tested device and its signature from the signature database completely in order to understand and better observe the performance of our system. Hence, in this case, GTID does not have the necessary master signature associated with a sample device/type tested. As a result, if GTID does not recognize a device, it then identifies the test device as an unknown device, otherwise it identifies the type and/or the device. Note that the existing studies do not seek to detect unknown devices (devices where they do not have a signature) as discussed in Section 2. Indeed, GTID's capability to operate in both test modes (*known* and *unknown*) makes it a comprehensive approach as the two different modes work best for different scenarios. For example, in a benign network, the *known* mode can be used for inventory control. However, in a network where access control is a concern, GTID can be employed in *unknown* mode in addition to other security solutions. Using the accuracy (α) and recall (γ) metrics explained in Section 5.3, we analyzed the overall effectiveness of our technique for these different modes. Nonetheless, as noted earlier, to analyze the performance specific to device IDs and device types, we only focus on γ because α is inflated superfluously by TNs. We 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, we include results pertinent to both metrics.

6.1 Experiments on the Isolated Wireless Testbed

Initially, we experimented GTID on an isolated wireless environment (Fig. 8a). Conducting experiments in an isolated setup allowed us to get a more 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?*

As seen in Table 4, the device with the maximum γ is *Netbook #5* with 100 percent and the average is 94 percent for the device identification analysis in the known analysis mode. For the same devices, in the unknown mode, the maximum and the average fall to 93 and 81 percent. The maximum and the average recall values for both known and unknown test modes for device type identification are lower compared to that of device identification. Nonetheless, we show through experiments in Section 6.2 that recall of device type identification can be significantly improved by using samples from more devices of the same type for training (Fig. 10a). As for the protocols and applications evaluated, UDP at the rate of 1 Mbps has a recall of 98 percent (known) and 83 percent (unknown) for device identification and 56 Byte Ping has a recall of 92 percent (known) and 90 percent (unknown) for device type identification. In comparison with UDP, a slight performance penalty with SCP was observed that is attributed to the reactive nature of TCP to round trip times and congestion.

6.2 Results from Experiments on the Wireless Campus Network

We conducted experiments in a live network to determine the feasibility of the technique and provided bounds for the

TABLE 4
Isolated Network—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	-	0.92	*iPhone4*	-	0.99	-	0.78
	Traffic Type					**Traffic Type**				
Max	*UDP 56B 1Mbps*	1.00	0.98	0.97	0.80	*Ping 56B*	0.97	0.92	0.90	0.74
Avg		0.99	0.94	0.97	0.78		0.87	0.67	0.83	0.54
	95% Confidence	$\pm$ 0.006	$\pm$ 0.04	$\pm$ 0.01	$\pm$ 0.04		$\pm$ 0.07	$\pm$ 0.29	$\pm$ 0.05	$\pm$ 0.2

performance of our technique in realistic deployments. Therefore, in the campus network, we were specifically interested in answering the following question: *What is the overall accuracy (α) and recall (γ) of this technique in a campus network?* General results for the campus network testbed are summarized in Table 5.

As seen in Table 5, in the campus network testbed the device with the maximum γ is *Kindle* #2 with 93 percent and the average is 74 percent for the device identification analysis in the known operational mode. In the unknown operational mode, the maximum and the average γ for these devices fall to 85 and 64 percent respectively. Similar to what was observed in the isolated experiments, the maximum and the average recall values for both known and unknown operational modes for device type identification is smaller compared to device identification.

As mentioned in Section 6.1, the performance of device type identification operational mode can be significantly improved by using more training devices for each type of device. Fig. 10a shows the results of an experiment that illustrates the improved recall value as more training devices are considered. In this experiment, we start by training on one representative device for each device type and continue to increase the number of representative devices used to train the Asus netbooks (device type). From the results shown in Fig. 10a, we clearly see that the recall of the Asus netbook (device type) increases as the number of training devices increases for each traffic type. As for the applications and protocols, the maximum γ and α are different for device and device type identification experiments. Specifically, for the device identification experiments, UDP with a rate of 8 Mbps and 1400 B payload exhibits the maximum γ: 80 percent; while for the device type experiments, the maximum is achieved by Skype traffic with γ: 96 percent. In the future, we plan to determine why the recall values of these signatures are traffic and device type dependent.

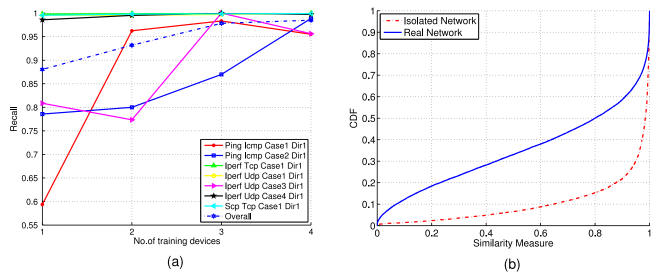


Fig. 10. (a) Effect of increase in training data (b) Effect of MAC contention.

Comparing the campus network to the isolated network experiments, we see that the performance degrades across both α and γ in the campus network. Across all the cases, we also observe that device identification results are better than the device type identification. As illustrated in Fig. 10a, this situation may possibly be improved with more devices to train for each device type, which will be explored more exhaustively as part of our future work.

We further analyzed the impact of physical and MAC layer interference on the campus network as presented in Fig. 10b. Focusing on the similarity measures for these two testbeds, we see that more than 50 percent of the similarity measures have values close to 1 (which denotes a perfect match) in the isolated testbed, while it is less than 30 percent for the campus testbed. We attribute this difference to the uncontrolled characteristics of the physical medium which make inter arrival patterns look less similar in the campus network.

Finally, although the slight performance decrease associated with the unknown test mode observed in all the test scenarios is attributed to the nature of the identification algorithm, our ANNs-based identification technique shows strong promise for the effectiveness of GTID (given its numerous benefits described in Section 1).

6.3 Active Fingerprinting

GTID operates on a stream of packets from the target device in order to generate a signature using the IATs (the results of this approach are provided in 5). On a campus network, this might not always be possible. For example, a network administrator might want to identify a suspicious node on the network that does not communicate frequently. In such a scenario, it would be very useful to have an active fingerprinting technique that can fingerprint devices using the response generated by the target device. In this analysis, we continuously probe the target device with ping packets and apply GTID on the stream of ping responses that is generated. The pinging rate was fixed at 100 pings per second and we analyze GTID's performance for payload sizes of 56 bytes and 1,400 bytes (Table 1).

From the results (Table 6) obtained in this analysis, it can be observed that Kindle (similar to passive analysis results) has the highest recall of 91 percent, which shows that it is quite suitable for fingerprinting using GTID. It is also observed that HP printers have the highest recall for device type identification. This indicates that the printers possess signatures that are very different from other types of

TABLE 5
Campus Network - 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	-	0.88	*Asus Netbook*	-	0.99	-	0.99
	Test Type					**Test Type**				
Max	*UDP 1400B 8Mbps*	0.96	0.80	0.94	0.66	*Skype*	0.96	0.96	0.85	0.86
Avg		0.95	0.74	0.93	0.60		0.86	0.68	0.83	0.61
	95% Confidence	$\pm$ 0.01	$\pm$ 0.08	$\pm$ 0.02	$\pm$ 0.07		$\pm$ 0.06	$\pm$ 0.25	$\pm$ 0.05	$\pm$ 0.15

TABLE 6
Campus Network - 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
	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
Avg		0.97	0.69	0.96	0.55		0.94	0.74	0.94	0.70
	95% Confidence	$\pm$ 0.006	$\pm$ 0.07	$\pm$ 0.01	$\pm$ 0.08		$\pm$ 0.023	$\pm$ 0.16	$\pm$ 0.03	$\pm$ 0.14

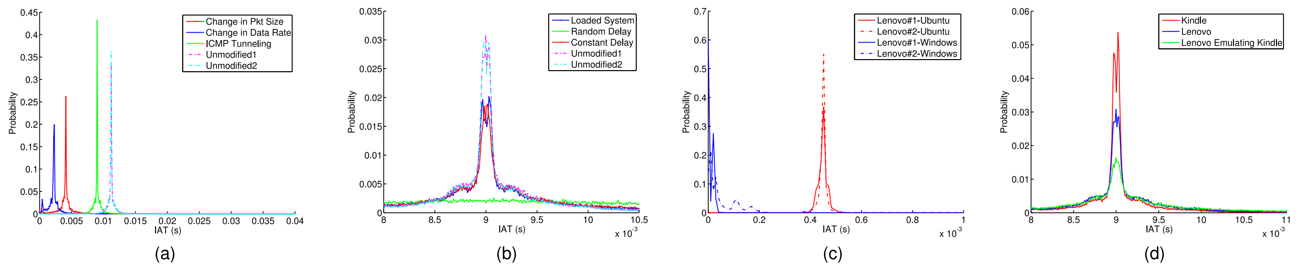


Fig. 11. Distribution of IATs for different attacker scenarios : (a) PDF of UDP IATs for normal and attacker traffic (b) PDF of ping response IATs for normal and attacker traffic (c) O/S attack (d) laptop emulates a Kindle.

devices, but quite similar to one another. From the per traffic type analysis, it can be seen that Ping with 1,400 byte payload does better than Ping with a 56 byte payload. This is consistent for both device and device type identification and also across both known and unknown analysis. The reason behind this is attributed to the fact that a continuous stream of large packets involves more memory operations, which in turn could provide a stronger fingerprint of the devices' hardware. In general, the active mode of fingerprinting performs slightly worse compared to the passive analysis (Tables 5 and 6). The reason is, in active fingerprinting, the IAT patterns embed the signature of both the probing device and the target device. Investigating methods for removing the sender's signature from the IAT pattern will be a part of our future work.

6.4 Analysis of Attacker Models

In this section, we seek to determine the effectiveness of GTID under various attack scenarios. Assuming that attackers are knowledgable about GTID and given that GTID is IAT-based, eight unique attackers, as articulated in Section 4 were considered. These attacks were implemented and tested against GTID.

Fig. 11a shows distributions of traffic where an attacker can vary the packet sizes, change the data rate and tunnel packets through another protocol. Fig. 11b presents distributions where an attacker has introduced constant/random delays to the packet stream and loaded the CPU with intensive applications to over shadow normal behavior. Fig. 11c shows distributions where an attacker has changed the node of interest's operating system. When we run these attacks on GTID, it detects and classifies all of them to be coming from an *unknown* source even though they were performed from a known devices. This outcome can be seen as an alert by the network administrator, who can take necessary actions. The variation in the IAT distribution patterns (from normal) observed in Fig. 11 explains why GTID was able to identify the attacker traffic.

However, a highly skilled attacker 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 IAT pattern of his device and the device that he wants to emulate. Once this information is in hand, he can use 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

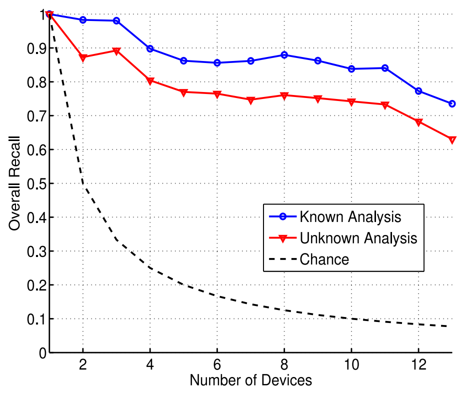


Fig. 12. Scalability analysis.

an attack is perpetrated, one would expect the attacker's device to be classified as a known device. However, as seen in Fig. 11d this is not the case. Fig. 11d 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 legitimate distribution and GTID labels the device that generated 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 prevent 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. 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.

7 Analysis of GTID's Limitations

While it is important to understand the performance of a detection system and its advantages, it is also equally important to understand its draw backs. Getting such clarity into both the positives and negatives of a technique is very important for system designers to wisely pick and integrate techniques that can work together in providing a complete security solution. In this section, we aim at providing a better understanding of GTID's limitations through experimental analysis and discussions.

7.1 Scalability

GTID is able to identify wireless devices and their device types by simply monitoring traffic generated by the wireless devices. The numerous benefits of such an approach are enumerated in Section 1. However, as with all techniques, GTID has limitations. Fig. 12 shows the average recall of GTID demonstrated on a live campus network as the number of devices being fingerprinted is increased. The figure also illustrates the likelihood (without a system like GTID) of detecting a device without expensive spectrum analyzers covering the entire wireless network and without client software installed on a node (i.e., chance). The figure shows the recall of GTID drops off fairly linearly with increase in the number of nodes. Although GTID may not be a general solution for every network, as discussed in Section 1, GTID could help secure 87 percent of the current corporate networks.

7.2 Performance Analysis with Increased Accuracy in Time-Stamping

Given that the performance of GTID is completely dependent on the amount of hardware signature that gets embedded into the IATs, it is imperative that we preserve it well. For all the analysis discussed above, we used a normal ethernet interface card plugged into a desktop computer for monitoring the network traffic. The time-stamping in such a setup provides accuracy only up to micro seconds. In addition to this, since time-stampings are done at the kernel, the accuracy of IATs can go down even further. In order to study the impact of timestamp accuracy on the performance of GTID, we repeated the wireless experiment for SCP traffic type using a 10 Gig NetFPGA to capture the packets. The hardware/software combination allowed us to timestamp the packets with nanosecond accuracy, and importantly, the time-stamping was done in hardware and not at the kernel level.

Surprisingly, this experiment did not show any improvement compared to the results achieved using a normal NIC. This attributes to the fact that the magnitude of timing noise added by the CSMA/CA protocol and the switches on the network might be in the order of microseconds, which makes nanosecond time-stamping less useful. Another problem with using highly accurate timestamps is that it requires very small bin sizes in order to capture variations that are happening at the nanosecond scale. This results in a huge increase in the size of the input vector passed on to the ANNs, which in turn makes the neural network more complex and slow. This experiment shows that the analysis results discussed in other sections are the best achievable results from the stand point of timestamp accuracy. In order to further improve the performance of GTID, we need to look into other alternative areas, which is a part of our future work.

8 Real-Time Implementation

Detection techniques need to be quick in order to be of any practical use. High accuracy with high latency will 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 evaluated. The internal architecture of the prototype is shown in Fig. 13.

8.1 Overview

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

Stage 1: Network traffic collection. Before starting to collect traffic from the network interface, GTID is configured with the necessary parameters such as Sample Size (S), Number of Samples (N), and unknown thresholds. *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 they are created. This forms a small pipeline and thereby contributes to speed up of GTID. Another feature supported by the system is processing of existing pcap files. When this option is used, the system parses a pcap file that is given as

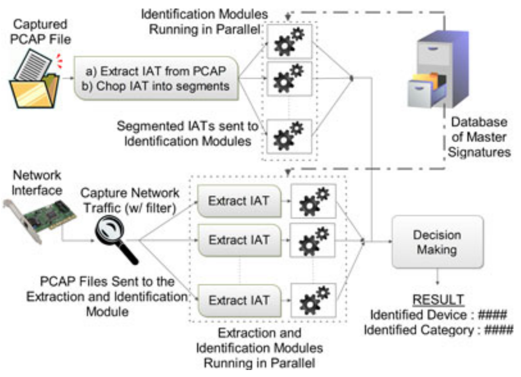

Captured PCAP File
Identification Modules Running in Parallel
a) Extract IAT from PCAP
b) Chop IAT into segments
Segmented IATs sent to Identification Modules
Database of Master Signatures
Network Interface
Capture Network Traffic (w/ filter)
Extract IAT
Extract IAT
Extract IAT
Decision Making
PCAP Files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

Fig. 13. Design of real-time implementation.

input 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. This is useful for forensic analysis.

Stage 2: Identification. At this stage, once a pcap sample arrives, IAT values are extracted using matshark [21]. 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 the input traffic, it provides 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, each sample is assigned to a separate core or hyper thread of the CPU, using n matlabpool workers and parallel loops. If this stage receives sampled IAT matrices, then it skips the extraction block and directly enters into the process of identification as shown in Fig. 13. This happens when pcap files are fed into stage one of the process.

Stage 3: Decision making. In the decision making stage, the system 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 matched most of the input samples and declares that device/type as the final identified device/type.

The system also has a graphical user interface which makes it easy to enter the input values and display results. It 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 are in the database which will in turn help in speeding up the identification process. This is useful if one has an idea of the types of unknown devices on the network.

8.2 Performance Analysis

In order to quantify the performance of our technique, we developed a prototype of GTID. The two most important factors that delay the identification process are *capture time* and *processing time*. The capture time is dependent on the sample size and the processing time is dependent on the algorithm and processing power. Table 7 shows the variation in the time taken to identify a device when the sample sizes are varied. The time taken to capture packets 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 percent as sample size is increased from 1 to 50 K.

TABLE 7
Timing Analysis for Variation in Sample Sizes : Traffic—UDP 1,400byte 8 Mbps, 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

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 7. 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 8 shows the processing time when GTID is 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 percent when parallelized.

9 Conclusion and Future Work

In this paper, we introduced GTID, a technique which can operate actively or passively to fingerprint wireless devices and their types. GTID exploits the heterogeneity of devices, which is a function of the different device hardware compositions and variation in device clock skew. We applied this technique within the realm of 802.11 networks. We demonstrated the effectiveness and the practicality of GTID on both an isolated testbed and a live campus network using artificial neural networks. Further, we showed, using a traffic capture of over 400 hours and 300 GB emanating from 37 different devices (e.g., iPads, iPhones, Kindles, GooglePhones, Netbooks, etc.) with a diverse set of operating systems and traffic types (e.g., Skype, ICMP, SCP, Iperf), that GTID achieves high accuracy and acceptable recall in

TABLE 8
Effect of Parallelizing: Sample Size 2,500, 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

identifying previously seen and unknown devices and device types. We also addressed the efficacy of GTID under eight different attack models; provided algorithmic provision for device, device type and unknown signature detection; and conducted performance analysis using a real-time prototype implementation that involved multiple types of hardware and traffic configurations. With its promising results, GTID can complement existing security solutions such as those that provide authentication, network management, and access control systems.

In the future, we are generally interested in understanding the long-term stability of the signature. We plan to study the performance of GTID when device and device type identification is conducted over various access links (e.g., DSL, LTE) and under different temperature. Additionally, we plan to improve the robustness of GTID so it can better handle congestion on a link and varying levels of load on a node. Further, we plan to extend the application of GTID to provide remote detection of resource utilization on a node.

Acknowledgments

The authors thank Supreeth Sathyanarayana for useful experiments in supporting our background discussion. This work was supported partially by NSF-CAREER-CNS-0545667 844144 and DARPA-N10AP20022 grants.

References

[1] T. Kohno, A. Broido, and K. C. Claffy, "Remote physical device fingerprinting," in *Proc. IEEE Symp. Security Privacy*, 2005, pp. 211–225.

[2] S. Jana and S. K. Kasera, "On fast and accurate detection of unauthorized wireless access points using clock skews," in *Proc. ACM Int. Conf. Mobile Comput. Netw.*, 2008, pp. 104–115.

[3] F. Lanze, A. Panchenko, B. Braatz, and A. Zinnen, "Clock skew based remote device fingerprinting demystified," in *Proc. IEEE Global Commun. Conf.*, 2012, pp. 813–819.

[4] C. Neumann, O. Heen, and S. Onno, "An empirical study of passive 802.11 device fingerprinting," in *Proc. Int. Conf. Distrib. Comput. Syst. Workshops*, Jun. 2012, pp. 593–602.

[5] V. Brik, S. Banerjee, M. Gruteser, and S. Oh, " Wireless device identification with radiometric signatures," in *Proc. ACM 14th Int. Conf. Mobile Comput. Netw.*, 2008, pp. 116–127.

[6] J. Hall, M. Barbeau, and E. Kranakis, "Rogue devices in bluetooth networks using radio frequency fingerprinting," in *Proc. IASTED Int. Conf. Commun. Comput. Netw.*, 2006, pp. 108–113.

[7] K. Gao, C. Corbett, and R. A. Beyah, "A passive approach to wireless device fingerprinting," in *Proc. Int. Conf. IEEE/IFIP Dependable Syst. Netw.*, 2010, pp. 383–392.

[8] S. Bratus, C. Cornelius, D. Kotz, and D. Peebles, "Active behavioral fingerprinting of wireless devices," in *Proc. ACM WiSec Conf.*, 2008, pp. 56–61.

[9] J. Francois, H. Abdelnur, R. State, and O. Festor, "Ptf: Passive temporal fingerprinting," in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manage.*, 2011, pp. 289–296.

[10] J. Francois, H. Abdelnur, R. State, O. Festor, "Machine learning techniques for passive network inventory," *IEEE Trans. Netw. Service Manage.*, vol. 7, no. 4, pp. 244–257, Dec. 2010.

[11] J. Francois, R. State, T. Engel, and O. Festor, "Enforcing security with behavioral fingerprinting," in *Proc. 7th Int. Conf. Netw. Serv. Manage.*, 2011, pp. 64–72.

[12] L. Letaw, J. Pletcher, and K. Butler, "Host identification via USB fingerprinting," in *Proc. 6th Int. Workshop Syst. Approaches Digit. Forensic Eng.*, 2011, pp. 1–9.

[13] F. Koushanfar, S. Fazzari, C. McCants, W. Bryson, P. Song, M. Sale, and M. Potkonjak, "Can EDA combat the rise of electronic counterfeiting?" in *Proc. ACM/EDAC/IEEE Des. Autom. Conf.*, 2012, pp. 133–138.

[14] S. Sathyanarayana, W. H. Robinson, and R. Beyah, "A network-based approach to counterfeit detection," in *Proc. IEEE Int. Conf. Technol. Homeland Security*, Nov. 2013, pp. 473–479.

[15] Census numbers [Online]. Available: http://www.census.gov/econ/smallbus.html, Sept. 2014.

[16] V. Pai, H.-Y. Kim, and S. Rixner, "Exploiting task-level concurrency in a programmable network interface," in *Proc. 9th ACM Symp. Principles Practice Parallel Programm.*, 2003, pp. 61–72.

[17] J. L. Hennessey and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 4th ed. San Mateo, CA, USA: Morgan Kaufmann, 2006.

[18] M. H. Hassoun, *Fundamentals of Artificial Neural Networks*. Denver, CO, USA: A Bradford Book, 1995.

[19] G. Kakavelakis, R. Beverly, and J. Young, "Auto-learning of SMTP TCP transport-layer features for spam and abusive message detection," in *Proc. 25th Int. Conf. Large Installation Syst. Admin.*, 2011, pp. 18–18.

[20] (2014). Gtid wireless device fingerprinting dataset [Online]. Available: http://crawdad.org/gatech/fingerprinting/

[21] Sharktools [Online]. Available: http://www.mit.edu/ armenb/sharktools/, Sept. 2014.

Sakthi Vignesh Radhakrishnan received the bachelor's degree in electronics and communication engineering from the SSN College of Engineering, Anna University, in 2011. In 2013, he graduated with a master's degree in ECE from Georgia Institute of Technology and joined Qualcomm Atheros as a firmware developer. During his master's at Georgia Tech, he was a member of the Communication Assurance and Performance. His research interests include network security and wireless networking. He is a student member of the IEEE.

A. Selcuk Uluagac received the MS degree in information security from the School of Computer Science at Georgia Tech and the MS degree in electrical and computer engineering from Carnegie Mellon in 2009 and 2002, respectively. He received the PhD degree with a concentration in information security and networking from the School of Electrical and Computer Engineering at Georgia Tech in 2010. He is currently an assistant professor in the Department of Electrical and Computer Engineering at Florida International University. He has served as a member of the research faculty as a senior research engineer in the School of ECE at Georgia Tech. Prior to Georgia Tech, he was a senior research engineer at Symantec. The focus of his research is on cybersecurity with an emphasis on its practical and applied aspects.. He is a member of ASEE, ACM and a senior member of the IEEE.

Raheem Beyah received the bachelor's of science degree in electrical engineering from North Carolina A&T State University in 1998. He received the master's and PhD degrees in electrical and computer engineering from Georgia Tech in 1999 and 2003, respectively. He is an associate professor in the School of Electrical and Computer Engineering at Georgia Tech where he leads the Georgia Tech Communications Assurance and Performance Group and is a member of the Georgia Tech Communications Systems Center (CSC). He served as a guest editor for MONET. He is an associate editor of several journals including the (Wiley) *Wireless Communications and Mobile Computing Journal*. His research interests include network security, wireless networks, network traffic characterization and performance, and security visualization. He received the National Science Foundation CAREER Award in 2009 and was selected for DARPA's Computer Science Study Panel in 2010. He is a member of ASEE, a lifetime member of NSBE, and a senior member of ACM and IEEE.