A Passive Technique for Fingerprinting Wireless Devices with Wired-side Observations

A. Selcuk Uluagac*, Sakthi V. Radhakrishnan*, Cherita Corbett†, Antony Baca† and Raheem Beyah*
*GT CAP Group, School of ECE
Georgia Institute of Technology
Atlanta, GA 30332, USA
{seclcuk,sakthi03,rbeyah}@gatech.edu
†Johns Hopkins Applied Physics Lab
11100 Johns Hopkins Road
Laurel, Maryland 20723, USA
{cherita.corbett,antony.baca}@jhuapl.edu

***Abstract*—In this paper, we introduce GTID, a technique that passively fingerprints wireless devices and their types from the wired backbone. GTID exploits the heterogeneity of devices, which is a function of different device hardware compositions and variations in devices' clock skew. We use statistical techniques to create unique, reproducible device and device type signatures that represent time variant behavior in network traffic and use artificial neural networks (ANNs) to classify devices and device types. We demonstrate the efficacy of our technique on both an isolated testbed and a live campus network (during peak hours) using a corpus of 27 devices representing a wide range of device classes. We collected more than 100 GB of traffic captures for ANN training and classification. We assert that 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 the first 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 illustrate how it can be used to complement existing authentication systems and to detect counterfeit devices.**

***Index Terms*—GTID, Device Fingerprinting, Wireless Device Fingerprinting, Device Type Fingerprinting, Access Control**

I. Introduction

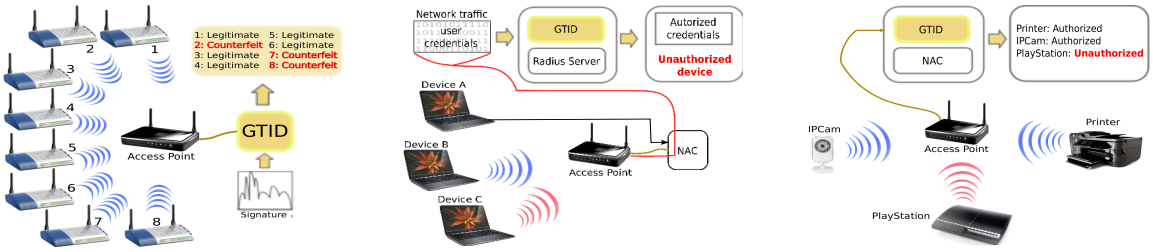
Identifying devices connected to a network (i.e., *device fingerprinting*) has become of critical importance to ensure, among other security services, access control to the network. In the same vein, there has also been a need to understand the type of a device that is connected to a network (i.e., *device 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: A Technique for Wireless Fingerprinting using Wired-side Observations*. 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. This heterogeneity is a function of the different hardware compositions (e.g., processor, DMA controller, memory) of the devices as well as the device's clock skew. The former enables device type identification and the latter enables device identification. We use statistical techniques to capture time-variant behavior in traffic and to create unique, reproducible device and device type signatures. 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 alleviates the need for costly spectrum analyzers and the need to be within wireless range of a device to be fingerprinted.

In general, current fingerprinting techniques either identify unique devices [1]–[6] or identify device types [7]–[12]. However, a technique that does both can prove very useful. For example, 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 (Figure 1(a))[1]. On the other hand, the device ID capability provided by GTID can be used to augment existing Network Access Control (NAC) systems that seek to control access to the network by using software installed on the node to pass the user's credentials to a backend server (e.g., RADIUS server) for authorization. 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 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 unauthorized, and access the network legitimately. With GTID used in conjunction with a NAC, the device can be authorized independent of the user's credentials (Figure 1(b)). 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 theormostats, gaming systems) as unmanageable. Since GTID uses information leaked from the architectural composition of devices in the network traffic as a signature and does not require a client on machines to be fingerprinted, GTID allows accountability for devices that are traditionally deemed unmanageable (Figure 1(c)). Note that as

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

978-1-4799-0895-0/13/$31.00 ©2013 IEEE



(a) (b) (c)

Fig. 1. (a) Counterfeit device detection; (b) Device and user authentication with NAC; (c) Authentication of unmanageable devices with NAC.

we approach the Internet of Things (IoTs), most devices will be considered unmanageable and will not work with current NAC solutions.

The contributions of GTID are as follows: It (1) is the first technique to provide device *and* device type fingerprinting; (2) is the first technique to illustrate resilience to various attacker types; (3) is the first technique to 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) ensures that devices are authorized, not just the device users (i.e., independent of user credentials); (9) does not require 3^{rd} party 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 artificial neural networks. Using a collection of 27 devices with a diverse set of operating systems, we evaluate the performance of GTID by considering accuracy, recall, and processing time. We also consider various attack models and the performance of a real-time implementation of GTID. GTID is a very promising technique to complement existing security systems and to detect counterfeit devices.

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 (and other local network activities, e.g., counterfeit detection). Although the testbed and components from a cost standpoint was significant ($40,000), the authors recognize that the performance of the system using a cohort of 27 devices is likely not representative of a network with 1000+ devices. However, given that most business are small businesses, and therefore most networks are small networks (87% of US companies have 19 or fewer employees while .34% of US companies have 500 or more employees) [14], 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 II. In Section III, we discuss the background and theory behind this technique. Threat model and assumptions are articulated in Section IV. Section V provides an overview of the technique. In Section VI, we evaluate GTID while considering various attacker models and then move on to discuss about a real-time implementation of our technique. We conclude the paper and discuss future work in Section VII.

II. Related Work

The existing work in this area can be placed into two categories.

The first category of work focuses on fingerprinting device types. In [7], the authors fingerprint wireless AP types by actively probing them with various regular and malformed packets. This technique is limited to fingerprinting types of wireless AP types, whereas our scheme can be extended to any IP-enabled device (e.g., wired and wireless) and can fingerprint devices as well as device types. In [8], 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. 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. In their early work [10], their technique required knowledge of the entire syntax of the protocol. However, 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.

Another body of work that is relevant to the proposed work deals with fingerprinting specific hosts (i.e., device fingerprinting). In [1], a method for 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. In contrast to the work in [1], our work is independent of protocol (i.e., it works for TCP, UDP, and 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 the same lines as [2] (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 and not with wire side observations. 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.

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.

III. Background Discussion

Packet creation in a device is a complex process. It involves many internal parts of the device working together. 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 (MAC) unit [15].

Assuming that the effect of the OS can be abstracted, we 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. At the component level, inherent variations in the clock skews between devices [16] help make the packet creation process unique.

IV. Threat Model & Assumptions

As shown in Figure 2, in our 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 timing traffic from captured packets on a wired segment between the AP and the final destination to identify the wireless devices that are transmitting[2].

Since our technique is based on timing analysis (Section V), 8 different classes of attackers are considered. The first seven attackers are *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)

[2]Note that our technique can also be used to identify wired devices that are connected to the wired network.

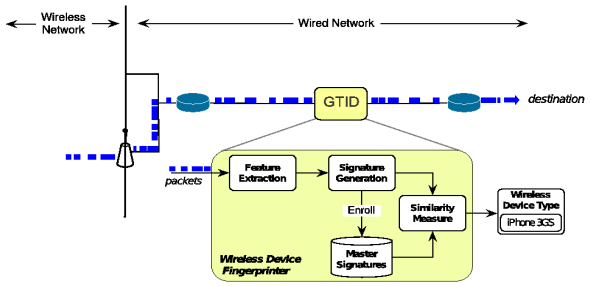


Fig. 2. Overview of GTID

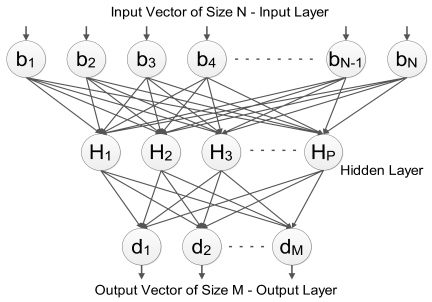


Fig. 3. Sample Neural Network

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 in order to establish/maintain network access. Analysis of these 8 skillful attackers and how GTID is resilient against these different threats are further discussed in Section VI-C.

V. Overview of GTID

In this section, we introduce the major components of GTID and discuss the machine learning based algorithm used to identify devices and their types. Then, we articulate the metrics in evaluating the overall performance of the technique.

A. Components of the Technique

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

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

$$f = (\Delta_{t_1}, \Delta\ t_2, , \Delta\ t_3, ..., \Delta\ t_i) \quad (1)$$

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 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].

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. We 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. 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.

Figure 3 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

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

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 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.

B. 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 (Kindle). Hence, given a set of master signatures, there would be three applicable outcomes in identifying a device and its type. 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, 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 each of the Kindle (device), and Kindle as a device type, along with the cutoff TPs X_{dev} and X_{cat} is shown in Figure 4. From the TP distributions in Figure 4, it can be observed that device TPs (red line) are more often closer to 1, compared to the device type TPs (green line). The clearly observed difference in the distribution patterns of device TP, the device type TP and TN from other devices (Figure 4) 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.

Algorithm 1 Device ID and Type Identification

```
1: Identify − ID − Type()
2: begin
3:   Θ_ID, Θ_Type, Dev_list, Type_list ← MastersDB(Traffic Type)
4:   λ_ID, λ_Type ← MaskingVectors()
5:   S ← IAT_Sample()
6:   Ω ← generate_signature(S)
7:   out_ID ← λ_ID * sim(Θ_ID, Ω)
8:   index, closeness ← max(out_ID)
9:   X ← 10_percentile_TP(Dev_list^index)
10:  if (closeness > X)
11:      return Dev_list^index, Corresponding Type
12:  else
13:    out_Type ← λ_Type * sim(Θ_Type, Ω)
14:    index, closeness ← max(out_Type)
15:    X ← 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
```

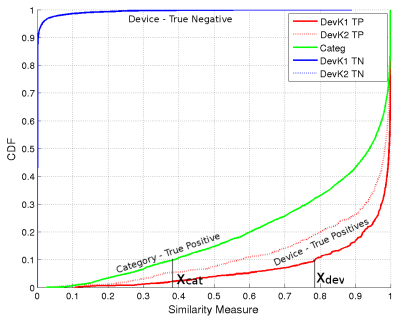


Fig. 4. CDF plot of closeness value for kindle fire - TCP traffic

C. Metrics for GTID's Effectiveness

We evaluate the performance of GTID using *accuracy* and *recall* as our metrics similar to [17]. 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 system. Recall is the measure of identifying an actual device and it is statistically defined as:

$$\gamma = \frac{TP}{TP + FN}. \quad (3)$$

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.

TABLE I
DEVICES USED

Isolated Testbed		Real Testbed	
Device Type	**Qty**	**Device Type**	**Qty**
Nokia N900	5	Lenovo G570 (Laptop)	2
Dell Latitude 2110 (Netbook)	2	Asus EeeePC 1025C (Netbook)	5
iPhone3G	2	Asus TF 101 (Tablet)	2
iPhone4G	2	Google Nexus One	2
iPad	3	Kindle Fire 1st Gen	2
Total	**14**	**Total**	**15**

VI. PERFORMANCE EVALUATION

In this section, we evaluate the performance of GTID across four dimensions. First, we analyze our technique in an isolated network environment (Figure 5(a)). Second, we measure the performance of GTID in a live campus network (Figure 5(b)) 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.

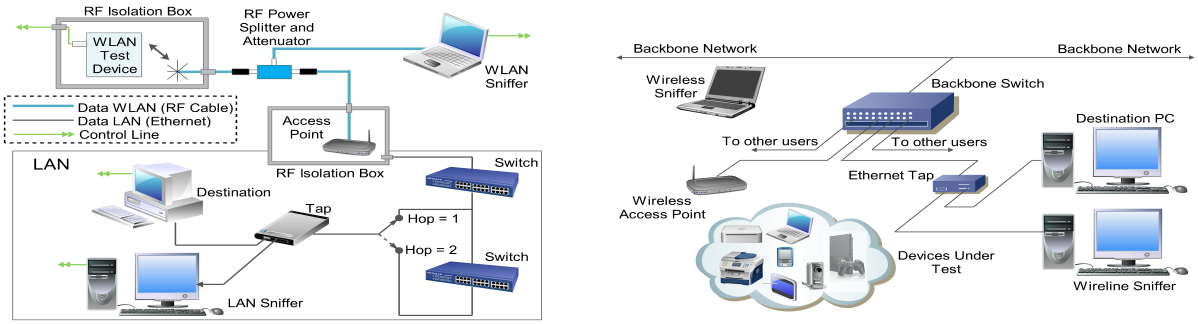
In GTID, two automated testbeds were assembled to transmit and record traffic from the wireless devices to the wired segment and vice versa. In the isolated testbed shown in Figure 5(a), a control machine (not shown in the figure) was used to send commands to the different devices in the testbed for 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 5(b)), 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. A total of 27 different devices, with an approximate total worth of $40,000 (14 in the isolated testbed, 15 in the real network testbed) were tested and the details which are listed in Tables I.

Furthemore, 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*. 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 the rate to *100 pings/second* and tested payload sizes of *64 Bytes* and *1400 Bytes*. For *UDP* analysis we used two payload sizes, *64 Bytes* and *1400 Bytes*, and sending rates of *1Mpbs* and *8Mbps*. We captured 1 hour of traffic for each of these traffic types, and this was done over 27 devices. This gave us a total of 319 hours (more than 100 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. We empirically determined the best sample sizes for a good *overall* performance of GTID to be 2.5K.

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*. 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. Note that the existing studies do not seek to detect unknown devices (devices where they do not have a signature) as discussed in Section II. Indeed, this limitation of the existing studies does not exist in GTID and its 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. Using the accuracy (α) and recall (γ) metrics explained in Section V-C, we analyze 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.

A. Results from the Experiments on the Isolated Testbed

Initially, we tested the performance of GTID in an isolated network environment (Figure 5(a)). 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*



(a) Isolated Network (b) Campus Network

Fig. 5. Setup of Campus and Isolated Network Testbeds

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? (5) How does queuing affect the technique overall? For queuing, two scenarios were tested in our isolated network testbed: Single-hop and multi-hop. To understand the effect of queuing, we used the multi-hop configuration of the isolated testbed. For the single hop case, as seen in Table II, the device with the maximum γ is *Netbook #5* with 100% 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 and the average fall to 92% and 78%. The maximum and the average recall values for both known and unknown test modes for device types identification are lower compared to that of device identification. Nonetheless, we show through experiments in Section VI-B that recall of device type identification can be significantly improved by using samples from more devices of same type for training (Figure 6(a)). As for the protocols and applications tested, UDP at the rate of 1Mbps shows a recall of 98% (known) and 80% (unknown) for device identification and 56Byte Ping shows a recall of 92% (known) and 74% (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. We also tested GTID on a multi-hop isolated scenario and found the average values of α and γ to be similar to the single-hop case[4]. This illustrates the feasibility of GTID for limited multiple-hop scenarios.

B. Results from Experiments on the Campus Network

We conducted experiments in a live network to determine the feasibility of the technique and provided bounds for the performance of our technique in realistic deployments. Therefore, in the real network, we were specifically interested in answering the following question: *What is the overall accuracy (α) and recall (γ) of this technique in a real network?* General results for the real network testbed are summarized in Table III.

[4]Note that the results from multi-hop experiments are not presented in this paper due to space constraints.

As seen in Table III, in the campus network testbed the device with the maximum γ is *Kindle #2* with 93% and the average is 74% for the device identification analysis in the known operational mode. In the unknown mode of these devices, the maximum and the average γ fall to 88% and 60%, respectively. Similar to what was observed in the isolated experiments, the maximum and the average recall values for both known and unknown test modes for device type identification is lesser compared to device identification.

As mentioned in Section VI-A, the performance of device type identification can be significantly improved by using more training samples for each type of device. Figure 6(a) shows the results of an experiment that we conducted to show if our hypothesis was true. In this experiment, we start by training on one representative device for each device type, and continue to increase in steps of 1, the number of representative devices used to train the Asus netbooks (device type). From the results (Figure 6(a)), we clearly see the recall of the Asus netbook (device type) increases as the number of training samples increase. As for the applications and protocols, the maximum γ and α different for device and device type identification 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%. In the future, we plan to determine why the recall values of these signatures are traffic and device type dependent.

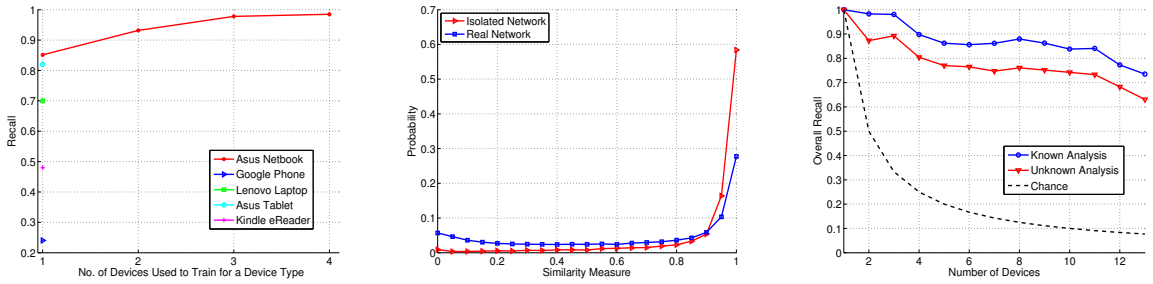
Comparing the real network to the isolated network experiments, we see that the performance degrades across both α and γ in the real network. Across all the cases, we also observe

TABLE II
ISOLATED NETWORK TESTBED

	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

TABLE III
CAMPUS NETWORK - WIRELESS

	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



(a) (b) (c)

Fig. 6. (a) Effect of Increasing Training Data; (b) Effect of MAC Contention; (c) Recall vs. Number of devices

.

that device identification results are better than the device type identification. As illustrated in Figure 6, this situation may possibly be increased 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 real network as presented in Figure 6(b). Focusing on the similarity measures for these two testbeds, we see 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. We attribute this difference to the uncontrolled characteristics of the physical medium which make inter arrival patterns look less similar in the real network. From these results, one can anticipate the results obtained in a wired network to be closer to the results obtained in the isolated testbed since there is no MAC or physical layer interference on a switched wired 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 I).

C. 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, 8 unique skillful attacker models were considered as articulated in Section IV and analyzed in this sub-section. Specifically, Figure 7(a) shows attackers that can vary their packet sizes, change their data rate, tunnel their packets through another protocol. Figure 7(b) presents attackers that can introduce constant/random delays to packet stream and load the CPU with intensive applications to over shadow normal behavior. Figure 7(c) shows an attacker that can modify/change its operating system. 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 7 explains why GTID was able to identify the attacker traffic.

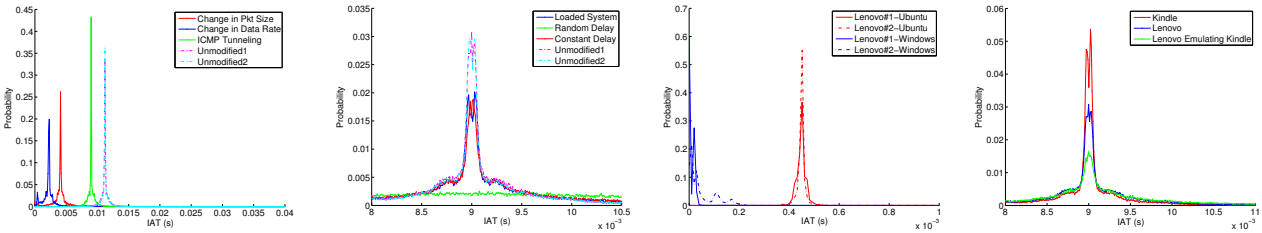
Furthermore, a further skillful attacker could try to emulate an authorized device in order to establish/maintain network access. To do this, the attacker would need the distribution of the difference in the IAT pattern of his device and the device that she wants to emulate. Once she has this information, 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 perpetrated, one would expect the attacker's device to be classified as a known device. However, as seen in Figure 7(d) this is not true. Figure 7(d) 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 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.

D. Scaleability

GTID is able to identify wireless devices and device types by simply monitoring traffic generated by the wireless devices on the wired side of the network. The numerous benefits of such an approach are enumerated in Section I. However, as with all techniques, GTID has limitations. Figure 6(c) shows the average recall of GTID demonstrated on a live campus network as the number of devices fingerprinted are 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 drop off is fairly linear with the increase in the number of nodes. Although GTID may not be a general solution for every network, as discussed in Section I, GTID could help secure 87% of the current networks.

E. Real-Time Implementation Discussion

A fingerprinting technique needs to be quick in order to be of any practical use. To study the detection speed of this technique, a prototype version of GTID was developed in



(a) (b) (c) (d)

Fig. 7. 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

MATLAB and its performance was tested. The two most important factors that affect time associated with GTID's decision process are *capture time* and *processing time*. The capture time depends on the data rate and number of packets needed (i.e., sample size) while the processing time is dependent on the algorithm and processing power. The time taken to perform the packet capture increases linearly with an increase in sample size. However the processing time increases at a slower rate. For example, when considering a sample size of 1K packets, the capture time for an 8Mbps UDP flow is 1.18s and the corresponding processing time is .0051s or 4.14% of the capture time. If the capture size increases to 50K packets, the corresponding processing time is 1.18s or 1.58% of the capture time. The capture size used in this work was 2.5K packets, which has a capture time of 3.95s and a corresponding processing time of .12s or 2.95% of the capture time. Thus, GTID is very lightweight (e.g., .12s to make a decision) as a single-threaded application when the bottleneck is the capture time. At higher data rates, the processing time will start to become the performance bottleneck. To overcome this, we implemented a multi-threaded version of GTID to process arriving packets in parallel. For example, when the data rate is 8Mbps, for a sample size of 2.5K packets, the total processing time jumps to 11.49s if GTID is single-threaded. But, if 8 threads are used the time is reduced to 6.67s, resulting in a 45% speed up by making GTID multi-threaded.

VII. Conclusion & Future Work

In this paper, we introduced GTID, a passive technique for fingerprinting of wireless devices and their types. GTID exploits the heterogeneity of devices, which is a function of the different device hardware compositions and the inherent variation in the chip fabrication process. We applied this technique to the challenging problem of access control in 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 collection of 27 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. We also addressed the efficacy of GTID under various attack models and considered the performance of a real-time implementation of GTID. We plan to understand the efficacy of GTID when device and device type identification is conducted over various access links (e.g., DSL, LTE). Additionally, we plan to improve the robustness of GTID so it can better handle congestion on a link and various levels of load on a node. Further, we plan to extend the application of GTID to the detection of counterfeit devices and remote detection of resource utilization on a node.

VIII. Acknowledgments

This work was partly supported by NSF-CAREER-CNS-0545667 844144 and DARPA-N10AP20022.

References

[1] B. A. Kohno, Tadayoshi. 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.

[2] 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.

[3] F. Lanze, A. Panchenko, B. Braatz, and A. Zinnen, "Clock skew based remote device fingerprinting demystified," in *Proc. of 55th International IEEE Global Communication Conference (GLOBECOM)*, 2012.

[4] C. Neumann, O. Heen, and S. Onno, "An empirical study of passive 802.11 device fingerprinting," in *Distributed Computing Systems Workshops (ICDCSW), 2012 32nd International Conference on*, june 2012, pp. 593 –602.

[5] 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.

[6] 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.

[7] 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.

[8] C. L. C. Ke Gao and R. A. Beyah, "A passive approach to wireless device fingerprinting." IEEE/IFIP, 2010.

[9] J. Francois, H. Abdelnurt, R. State, and O. Festort, "Ptf: Passive temporal fingerprinting." IFIP/IEEE, 2011.

[10] ——, "Machine learning techniques for passive network inventory," vol. 7, 2010.

[11] J. Francois, T. Engel, R. State, and O. Festort, "Enforcing security with behavioral fingerprinting," 2011.

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

[13] F. Koushanfar, S. Fazzari, C. McCants, W. Bryson, M. Sale, P. Song, and M. Potkonjak, "Can eda combat the rise of electronic counterfeiting?" in *Proc. of 49th ACM/EDAC/IEEE Design Automation Conference*, 2012.

[14] "Census numbers," http://www.census.gov/econ/smallbus.html.

[15] H. Kim, V. Pai, and S. Rixner, "Exploiting task-level concurrency in a programmable network interface," in *Proc. of Ninth ACM SIGPLAN symposium on Principles and practice of parallel programming*, 2003.

[16] V. Paxson, "On calibrating measurements of packet transit times," in *In Proceedings of ACM SIGMETRICS*, 1998, pp. 11–21.

[17] G. Kakavelakis, R. Beverly, and J. Young, "Auto-learning of smtp tcp transport-layer features for spam and abusive message detection," in *LISA*, 2011, pp. 18–18.