Automated IoT Device Identification using Network Traffic

Ahmet Aksoy
Computer Science and Engineering
University of Nevada, Reno
Email: aksoy@nevada.unr.edu

Mehmet Hadi Gunes
Computer Science and Engineering
University of Nevada, Reno
Email: mgunes@unr.edu

Abstract*—IoT devices have been gaining popularity and become integral to our daily life. These devices are prone to be compromised as well as any computing system, but lack computing capabilities for cybersecurity software. An important measure for protecting attacks on IoT devices is through isolation of such devices by restriction of communications to the device from firewall/gateway. To this end identification of the IoT device is valuable for network administration and security. In this paper, we introduce a system for automated classification of device characteristics, called System IDentifier (SysID), based on their network traffic. SysID uses any single packet that is originated from the device to detect its kind. We use genetic algorithm (GA) to determine relevant features in different protocol headers and then deploy various machine learning (ML) algorithms (i.e., DecisionTable, J48 Decision Trees, OneR , and PART) to classify host device types by analyzing features selected by GA. GA helps reduce classification complexity and increases its accuracy by eliminating noisy features from the data. SysID allows the ability to have a completely automated way of classifying IoT devices using their TCP/IP packets without expert input for classification. In an experimental study with 23 IoT devices, SysID identified the device type from a single packet with over 95% accuracy.

***Index Terms*—Device fingerprinting, Genetic algorithm, Machine learning, Passive measurements**

I. Introduction

Network management requires knowledge of devices attached to a network. A vulnerable device connected to the network could be utilized for insider attacks. Hence, network administrators identify devices that are connected to the network in order to monitor and secure the network [1]. As various Internet of Things (IoT) devices are introduced into a network, the network administrators need a better understanding of the devices that are connected. In particular, IoT devices lack computational capabilities of typical networked devices, and hence need closer monitoring to prevent/detect intrusion/malware. Securing IoT devices is not just important for the device itself, but also for other network components. For instance, Mirai botnet utilized over 400,000 IoT devices in launching the first DDoS attack with over 1 TB traffic volume [2].

Network administrators can automate the process of identifying attached devices using device fingerprinting [3]. Device fingerprinting is the process of remotely detecting the kind of device from its network traffic [4]. Active fingerprinting probes the system and analyzes the response it receives. Passive fingerprinting, on the other hand, sniffs packets generated by the system and analyzes the information extracted from the network traffic.

In this paper, we present an IoT device fingerprinting system System IDentifier (SysID) that can identify the device type from a single packet with high accuracy. We utilize machine learning and genetic algorithms to learn unique features of each IoT device without expert supervision. SysID is applicable to both active and passive fingerprinting. However, the data considered in this paper was passively collected, and hence the discussion is for passive fingerprinting.

Header field information in different protocols contain unique signatures and provide clues for classification of devices [5]. We utilize GA to select features that are most unique to the IoT device compared to other devices in the training data. In our analysis, depending on the packet content, we observed existence of 212 features which means 2^{212} unique combinations could be considered for classification. With fewer number of features to be considered, a lower classifier complexity would be possible. GA reduces the number of selected features to improve system efficiency in a large search space along with classification accuracy.

In an earlier study, we analyzed the contribution of various machine learning algorithms and identified J48, DecisionTable, OneR, and PART, among the best algorithms for Operating System classification using network traffic [4]. For every classifier that we build, we analyze the performance of all of these algorithms and select the one with the highest performance to ensure that the overall classification of SysID is at its maximum. Rule-based algorithms such as PART and J48 also allows us to analyze the features and their contributions to fingerprint IoT devices.

II. Related Work

Researchers have developed approaches to fingerprint devices using active probes or passive traffic captures. Nmap is an active OS fingerprinting tool which is also capable of device fingerprinting [6]. It takes advantage of different implementation of network stack by different vendors to detect the device type. Nmap generates up to 16 probes to detect the OS/device.

Couple of passive fingerprinting approaches focus on network packet features. P0f performs passive fingerprinting and leverages the TCP SYN packets' header information for OS detection [7]. Gao et. al. perform wavelet analysis on packet traffic to distinguish access points [8].

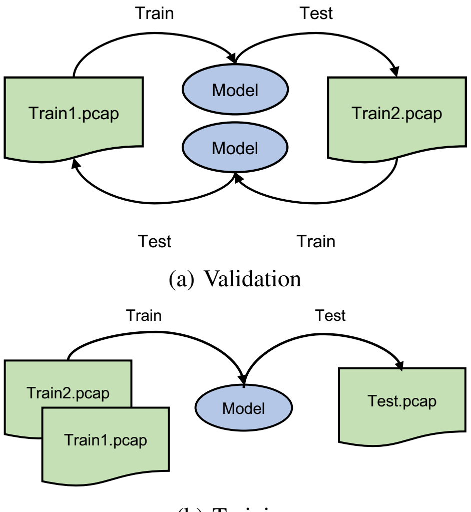
Several approaches focus on the timing characteristics. Passive and Temporal Fingerprinting performs device fingerprinting based on the application layer protocols' timing [9]. RTF represents fingerprints in a tree-based temporal finite state machine and uses SVM (Support Vector Machines) as classifier. Similarly, Radhakrishnan et. al. model the distribution of packet inter-arrival times (IAT) and use Artificial Neural Network (ANN) to classify devices [10]. Formby et. al. focus on identification of Industrial control systems with data response processing times and physical operation times as device signatures [11]. Kohno et. al. identify devices based on the distribution of clock skews, which are captured from TCP timestamps [12].

Couple of studies focus on the wireless network characteristics. Desmond et. al. detect devices connected to a Wireless-LAN by analyzing the timing of 802.11 probe request frames [13]. Authors employ clustering algorithms to generate fingerprints. Nguyen et. al. propose a passive fingerprinting technique using radio-metrics as the distinguishing feature of devices to detect identity spoofing [14]. The radiometrics include radio signal amplitude, frequency, etc. Authors then employ a non-parametric Bayesian method for device identification. Recently Xu et. al. focused on physical, MAC and upper layer characteristic to fingerprint wireless devices using a White-List Based Algorithm and Unsupervised Learning [15].

The most similar study to ours is [16], where authors propose an automated system for device fingerprinting. Miettinen et. al. collect n packets after a new IoT device initiates its setup phase. This n is determined when a decrease in the packet rate is observed. They use 23 features to fingerprint a device where 19 are binary values representing the absence or existence of the following protocols; link layer (i.e. ARP, LLC), network layer (i.e. IP, ICMP, ICMPv6, EAPoL), transport layer (i.e. TCP, UDP), application layer (i.e., HTTP, HTTPS, DHCP, BOOTP, SSDP, DNS, MDNS, NTP), data payload, and IP options (i.e. Padding, RouterAlert). Additional 4 features are integer values to count for packet size, destination IP counter, source port counter, and destination port counter. They remove consecutive identical packets. and use the first 12 unique vector packets and generate a 23x12 matrix as a fingerprint for every device. Then, using RandomForest algorithm, they generate a classifier for each device.

III. Device Fingerprinting from Network Traffic

In this paper, we present SysID, an entirely automated system to generate device fingerprints and classify IoT devices using a single TCP/IP packet. We use genetic algorithm (GA) for feature subset selection to determine the most unique packet header features for device fingerprinting among a large number of possible packet headers. To the best of our knowledge, our approach is the first to automate IoT device fingerprinting without expert input. SysID considers all network layer (i.e. IP and ICMP), transport layer (i.e. TCP and UDP) and application layer (e.g., DNS, HTTP, and SSL) protocols to determine the most unique protocol features among the protocols' header fields. SysID then runs a machine learning (ML) algorithm to determine the classification accuracy based on the set of features determined by GA.



(a) Validation

(b) Training

Fig. 1: Validation/Training

A. Feature Selection

We created 3 groups of pcap network captures for training, validating and testing as shown in Figure 1. In each group, we merge the pcap files into a single file and convert it to arff file format which is compatible to be run with WEKA tool. After this process, we end up with 3 arff files where 2 of them are used to train and validate the machine learning models, and the remaining one is used for testing. Let's call the arff files for training as Train1.arff and Train2.arff and the arff file for testing as Test.arff. Rather than only training with Train1.arff and validating it on Train2.arff, we want to make sure that the features that GA selected are general enough for our classifiers when either of these two sets of packets are used for building a model. This helps us adapt our classifier to possible unexpected cases in the testing data. As we realized that, in our case, classifiers would memorize, rather than learn [4], we did not perform 10-fold cross-validation.

We used a population size of 30 chromosomes in our GA implementation. A chromosome is a series of 0's and 1's, at the same length as the number of features in the arff file. If the arff file, for example, contained 100 features except for the class, then the chromosome's length would be 100. The values 0 and 1 for each bit of the chromosome determines whether to exclude or include a specific feature in machine learning. We have set GA to initially populate 30 chromosomes containing random 0's and 1's. For each of these chromosomes, GA runs the fitness function to determine how strong they are with machine learning. Depending on the fitness values that GA obtains for each of these chromosomes, it tries to converge to as optimal solution as possible by repeating the same process.

For our fitness calculation, we considered two metrics to determine the most suitable set of header features. Since we want to perform classification with as high performance as possible while ensuring that we use as small subset of features as possible, in our fitness function, we consider two metrics that represent these values. The fitness function returns a fitness value between 0 and 1 where 1 means the most contributing. As shown in the fitness function below, we used the weight of 0.9 for the accuracy and 0.1 for the reduction of the number of selected header fields.

$$\begin{aligned} Fitness = \; & 0.9 \times Accuracy + \\ & 0.1 \times \left(1 - \frac{|SelectedFeatures| - 1}{|AllFeatures| - 1}\right) \end{aligned}$$

If the number of features selected is equal to the number of features that exist in the dataset, the component returns 0. This indicates that, all header fields in the dataset are needed without any selection. However, if the number of features selected is 1, it indicates the contribution of this single header feature is sufficient for classification.

The accuracy in the fitness function is calculated through applying machine learning to the selected features in a chromosome. This term tells us how well the selected features in the chromosome classifies the testing dataset when the machine learning model is trained using these features. To calculate this component, we train an ML model on the Train1.arff dataset using only the features selected by the chromosome. Then, we test the model on Train2.arff and record the performance of classification. Additionally, we swap train datasets and train the model using Train2.arff using only the features selected by the chromosome to be tested on the Train1.arff. After getting performance results for both cases, we calculate the average to be used as the accuracy component in our fitness function.

The fitness function returns a value between 0 and 1 to tell GA how unique a particular chromosome is. GA tries to optimize this fitness value to converge to the best solution it can find which then helps us find a set of features that are used for classification.

Since it is not guaranteed for GA to converge to the optimum set of features yielding the highest fitness value, GA needs a termination point to prevent itself from going into an infinite loop of space exploration. In our case, we employed the decision of stopping the exploration after “n” consecutive occurrences of the same set of features being selected by the GA. If “n” is too low, it is possible for GA to quit before converging to a better possible set of features. If “n” is too high, depending on the implementation of the GA, it is possible for GA to either enter an infinite loop or take very long time to converge to a solution. We observed that “n=5” was sufficient enough for the termination of GA in our case. Additionally, it is possible for GA to converge to different sets of features in each run as GA starts with random initialization. Hence, rather than relying on the outcome of a single run, we run GA 10 times and select the features which produced the highest classification performance.

B. Classification

For classification of packets, we use machine learning algorithms in the WEKA tool [17]. WEKA provides a variety of machine learning libraries and we used DecisionTable and J48 Decision Trees, OneR, and PART. We observed in our previous work [4] that rule-based algorithms out-perform other approaches in classifying packets using the header fields. Since rules are generated based on the actual attribute values of the packets, algorithms could catch unique header features to classify devices. They also allow to observe the exact values of attributes used for classification. This helps us analyze the behavior of a device by investigating which features were used for identification and which values of features were distinctive between devices.

IV. Experimental Results

In this section, we present performance of System IDentifier (SysID) using 20 measurements from 23 IoT devices that include home sensors, coffee makers, power switches and lights.

A. Data

In this paper, we use a dataset collected for a similar study [16]. The dataset contains packets captured from numerous IoT devices. We used the data for 23 IoT devices that contained 20 measurements. The devices are: Aria, D-LinkCam, D-LinkDayCam, D-LinkDoorSensor, D-LinkHomeHub, D-LinkSensor, D-LinkSiren, D-LinkSwitch, D-LinkWaterSensor, EdimaxPlug1101W, EdimaxPlug2101W, EdnetGateway, HomeMaticPlug, HueBridge, HueSwitch, iKettle2, Lightify, MAXGateway, SmarterCoffee, TP-LinkPlugHS100, TP-LinkPlugHS110, WeMoLink, Withings. We removed 4 devices that were included in [16] as they had less than 20 measurements.

We separate the pcap network captures into training (14 measurements for each device, i.e., 70% of datasets) and testing (remaining 6 measurements, i.e., 30%) data. In each group, we merge the pcap files and extract all possible features for every packet in the pcap files. Then, we remove IP address and IP checksum (which correlates with IP address) to remove any bias that may occur in classifying the devices. We also remove string features such as *http.cookie* since they are not compatible with certain classifiers in WEKA. We then run GA to select relevant features with each machine learning algorithm (i.e., J48, DecisionTable, OneR, and PART) and record the algorithm with highest performance.

B. One-Level Classification

We first performed classification of all devices and tried to detect individual device types using SysID. Figure 2 presents

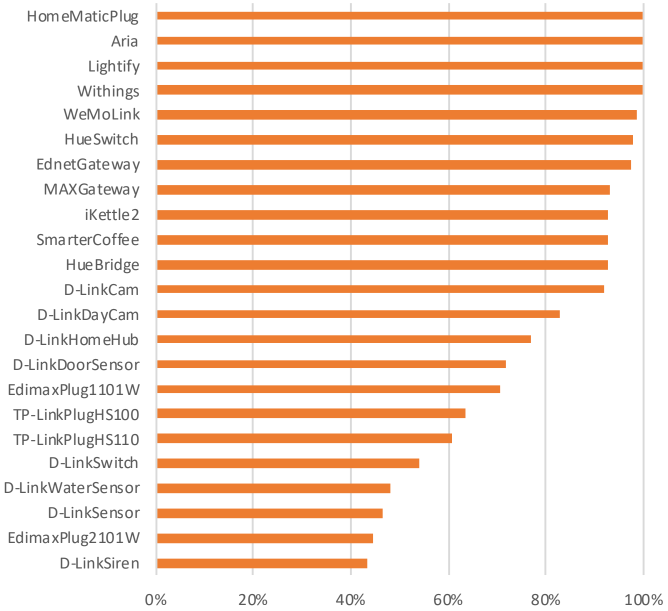


Fig. 2: Overall classification performance

the classification performance of a single packet for each device. As PART algorithm performed classification at higher rate than other algorithms, it was used in the classification of devices. Although numerous devices were classified correctly with high accuracy, some of the devices did not yield as high accuracy.

Furthermore, Table I presents the confusion matrix of SysID where rows show the actual device and columns show predicted classification. The optimal case would be when all packets are at the intersection of the same class both for actual and predicted classes. For example, for the “Withings” packets, 334 packets were classified as “Withings” but only 1 packet was classified as “EdimaxPlug2101W” which is an error. However, when we inspect “HueSwitch” and “HueBridge” packets, we observe that most of the packets were classified as either of these two devices. For “HueSwitch”, 10762 packets were correctly classified but 230 of the packets were classified as “HueBridge”. Similarly, for “HueBridge”, 7026 were correctly classified as “HueBridge” but 520 of the packets were incorrectly classified as “HueSwitch”. This indicates that the machine learning had difficulty distinguishing the behaviors of these two devices.

Overall, we observed that the machine learning had confusion in distinguishing the behaviors of several groups, namely {EdimaxPlug1101W, EdimaxPlug2101W}, {HueBridge, HueSwitch}, {TP-LinkPlugHS100, TP-LinkPlugHS110} and {D-LinkCam, D-LinkDayCam, D-LinkDoorSensor, D-LinkHomeHub, D-LinkSensor, D-LinkSiren, D-LinkSwitch, D-LinkWaterSensor}. These groups of devices are from the same vendors and hence most likely utilize similar network libraries. While {iKettle2, SmarterCoffee} belonged to the same vendor, their network packet headers were distinguishable.

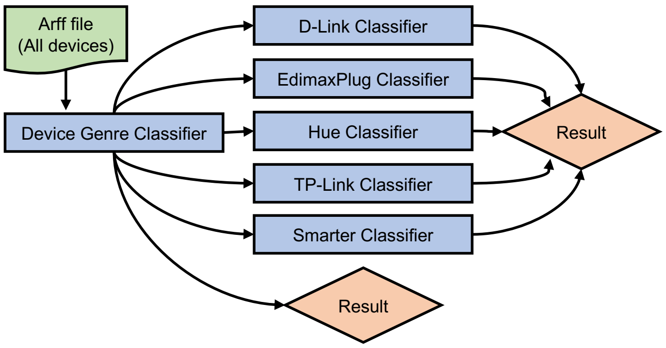


Fig. 3: Classifiers

C. *Two-level Classification*

As IoT devices from the same vendor were mixed, we decided to perform two-level classification where we first determine the vendor of devices and then differentiate devices from the same vendor. Hence, we generated 6 separate classifiers in a two level hierarchy as shown in Figure 3. The first classifier learns to classify either the device itself if there is only one device from the vendor (i.e. Aria, EdnetGateway, HomeMaticPlug, Lightify, MAXGateway, WeMoLink and Withings) or the device genre when multiple devices are by the same vendor (i.e. D-Link, EdimaxPlug, Hue, TP-LinkPlug and Smarter). If the packet’s class is determined to be one of the devices themselves, then that is the label that the classifier returns. If the packet is determined to belong to one of the genres, then the packet is forwarded to the corresponding classifier for further processing to detect the actual device. This hierarchical structure allows us to detect the best classification algorithm in each classifier, which helps improve overall classification performance. It is possible for different protocols and different machine learning algorithms to perform better for different packet sources.

Figure 4 presents the classification performances for each class in our dataset when the PART algorithm is used. Figure also shows the individual device classification performance of IoT Sentinel [16], marked with blue. The lowest classification performance is at 95% for MAXGateway. We observe SysID is able to obtain similar accuracy as IoT Sentinel, which uses 12 packet sequence in classification with expert input. As for the individual vendors with multiple devices, SysID can tell if a device is a Hue device at 99.8%, D-Link at 99.7%, EdimaxPlug at 98.9%, TP-LinkPlug at 96.2% and Smarter at 96.4%.

We then identify devices belonging to the same vendor. PART algorithm was able to differentiate between Hue devices with high accuracy as shown in Figure 5. SysID can determine if any individual packet originated from a HueSwitch with a rate of 98.8% and a HueBridge with a rate of 95.9%.

Similarly, PART algorithm provided best identification of TP-LinkPlug devices in Figure 6. We observe that SysID could determine TP-LinkPlugHS110 with an accuracy rate of 89.3% and TP-LinkPlugHS100 with 42.2% accuracy. We observe that the classifier is having difficulty in distinguishing these two

TABLE I: Confusion Matrix

Actual **Predicted**	Withings	Lightify	D-LinkSiren	Aria	D-LinkWaterSensor	D-LinkSensor	TP-LinkPlugHS100	MAXGateway	EdimaxPlug2101W	TP-LinkPlugHS110	D-LinkDayCam	D-LinkCam	D-LinkHomeHub	HueSwitch	SmarterCoffee	HueBridge	D-LinkDoorSensor	WeMoLink	HomeMaticPlug	iKettle2	EdnetGateway	EdimaxPlug1101W	D-LinkSwitch
Withings	334	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Lightify	0	1931	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
D-LinkSiren	0	0	1397	0	1199	348	5	0	0	3	0	1	121	1	0	0	8	4	0	0	0	0	121
Aria	0	0	0	237	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
D-LinkWaterSensor	0	0	727	0	1575	508	10	0	0	5	0	0	167	1	0	0	11	4	0	0	0	0	271
D-LinkSensor	0	0	1007	0	345	1589	7	0	0	7	0	1	106	1	0	1	7	3	0	0	0	0	336
TP-LinkPlugHS100	1	0	3	0	2	1	209	0	1	91	0	2	13	0	0	3	1	1	0	0	0	0	1
MAXGateway	0	7	0	0	0	0	0	308	0	0	0	0	0	16	0	0	0	0	0	0	0	0	0
EdimaxPlug2101W	0	0	1	0	0	17	3	2	125	2	2	0	0	0	0	1	0	0	0	0	0	128	0
TP-LinkPlugHS110	0	0	3	0	3	9	98	0	1	204	0	0	6	0	0	6	0	0	0	0	0	0	5
D-LinkDayCam	0	0	0	0	0	2	12	0	0	0	73	0	1	0	0	0	0	0	0	0	0	0	0
D-LinkCam	0	0	5	6	5	30	3	0	0	9	0	933	13	0	0	7	0	1	0	0	0	0	5
D-LinkHomeHub	0	0	175	0	106	204	7	0	0	16	0	10	3284	9	0	2	56	54	0	0	0	0	352
HueSwitch	0	0	1	0	1	0	0	0	0	0	0	0	1	10762	0	230	0	0	0	0	0	0	0
SmarterCoffee	0	1	0	0	0	0	0	0	0	0	0	0	0	0	13	0	0	0	0	0	0	0	0
HueBridge	0	0	0	0	0	6	0	1	0	13	0	0	2	520	0	7026	0	1	0	0	7	0	1
D-LinkDoorSensor	0	0	19	0	10	23	8	0	1	0	0	1	148	1	0	1	865	95	0	0	0	1	30
WeMoLink	0	10	0	0	6	2	1	0	2	0	0	0	2	0	0	8	5	2775	0	0	0	0	4
HomeMaticPlug	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	165	0	0	0	0
iKettle2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	13	0	0	0
EdnetGateway	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	5	0	0	0	0	294	0	0
EdimaxPlug1101W	0	0	0	0	0	15	5	0	53	1	0	0	0	2	0	2	1	0	0	0	0	191	0
D-LinkSwitch	0	0	310	0	146	393	7	0	0	7	0	4	649	1	0	0	32	13	0	0	0	0	1828

devices. Note that IoT Sentinel was not able to identify these devices with high accuracy either. One of these two devices have a higher classification accuracy due to the order of rules that the classifier uses.

Additionally, J48 algorithm provided best identification of EdimaxPlug devices in Figure 7. SysID can distinguish EdimaxPlug1101W at an accuracy rate of 70% and Edimax-Plug2101W at a rate of 72%. Although the classifier cannot make a perfect distinction between these two devices, it can still tell them apart at a rate of 70%, which is higher than one-level results of Section IV-B.

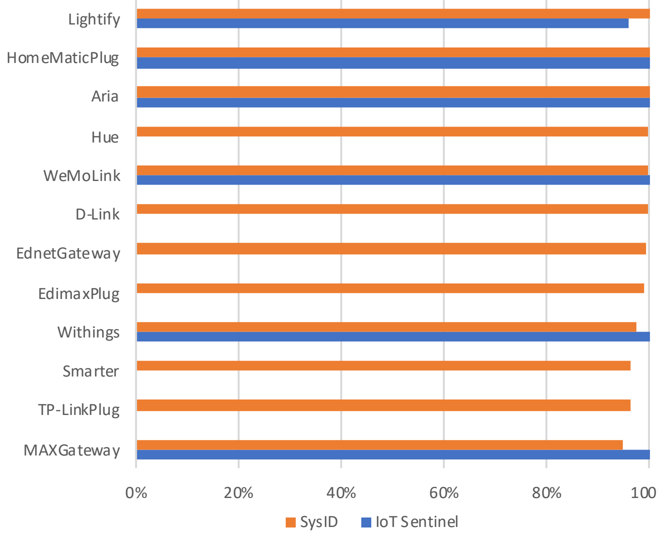


Fig. 4: Device genre classification performance

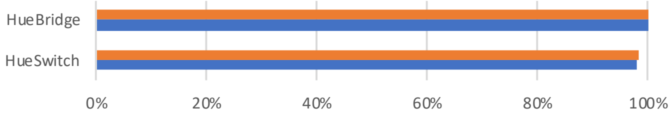


Fig. 5: Hue devices classification performance

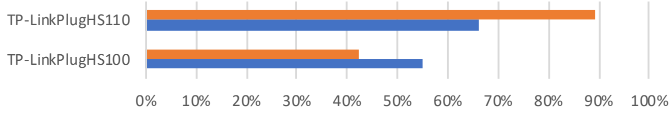


Fig. 6: TP-LinkPlug devices classification performance

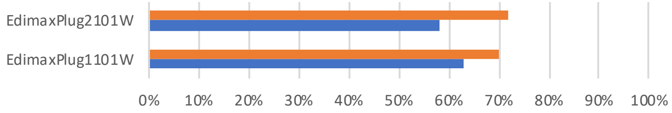


Fig. 7: EdimaxPlug devices classification performance

J48 algorithm also provided best identification of D-Link devices in Figure 8. SysID achieved identification accuracy of 94.2% for D-LinkCam, 86.4% for D-LinkDayCam, 84.5% for D-LinkHomeHub. However the remaining devices perform at lower classification rates indicating the possibility of similar single packet behavior between them. Note that IoT Sentinel had higher classification rate for D-LinkDayCam, D-LinkHomeHub, and D-LinkDoorSensor. The possible reason for this is that IoT Sentinel considers sequence of 12 packets, and this might be helpful to catch distinctive behavior of these devices.

Finally, J48 algorithm provided best identification of Smarter devices in Figure 9. SysID can determine if a packet

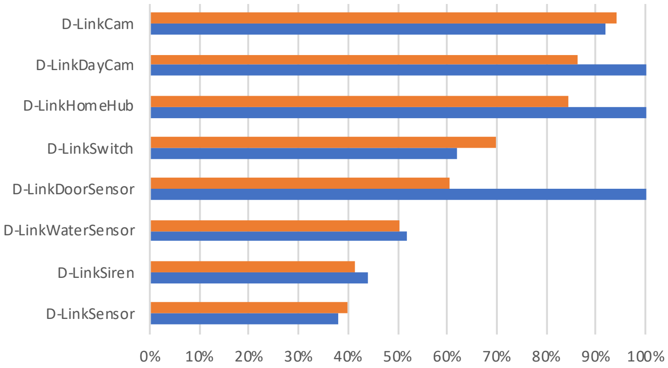


Fig. 8: D-Link devices classification performance

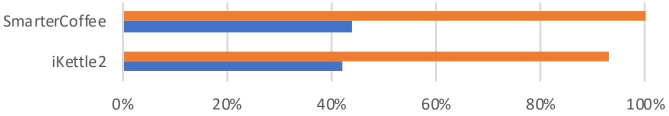


Fig. 9: Smarter devices classification performance

originated from an iKettle2 with an accuracy of 93% or from a SmarterCoffee with a rate of 100%. These results tells us that these two devices have very distinguishable behaviors which was not captured by IoT Sentinel.

D. Feature Subset Selection

SysID aims to increase fingerprinting accuracy by removing possible noisy data and decrease complexity by reducing the number of features selected using GA. Table II provides the number of features selected by the classifiers when GA is used. Except for Hue devices, using less than half of the features leads to better classification accuracy then when all of the features are used. SysID was able to fingerprint Smart devices at an average accuracy of 96% while using only 15% of the packet header features.

GA helps to automatically select a subset of features that contribute to the fingerprinting the most. However, for every classifier, the set of selected features differ. Table III presents the top 10 features selected by the classification algorithms for genre and each vendor classifier. We indicate whether a particular packet header feature was selected for each classifier. We observe that *udp.checksum* is the most preferred feature, as it was selected among top 10 by all the classifiers except for Hue devices. We believe *udp.checksum* is able to capture a unique periodic packet from these devices as UDP checksum is calculated over the data as well as the IP and UDP header fields.

TABLE II: Number of selected features

Classifier	Algorithm	GA	ALL	%
Device genre	PART	93	212	44%
Hue	J48	78	147	53%
TP-Link	PART	43	115	37%
EdimaxPlug	J48	51	123	41%
D-Link	J48	90	204	44%
Smarter	J48	8	55	15%

TABLE III: Selected Features

Features	All	Genre	Hue	TP-Link	EdimaxPlug	D-Link	Smarter
udp.checksum	✓	✓		✓	✓	✓	✓
dns.qry.class	✓	✓				✓	
ip.len	✓			✓		✓	
tcp.window_size	✓	✓	✓				
tcp.flags		✓	✓			✓	
icmp.checksum	✓	✓					
ip.dsfield	✓	✓					
dns.resp.len						✓	
ip.flags				✓			
ip.id		✓			✓		
ip.proto					✓		
ip.ttl	✓		✓			✓	
tcp.seq			✓			✓	
tcp.ack						✓	
tcp.options.timestamp.tsval			✓	✓	✓		
tcp.port			✓	✓	✓		
tcp.stream			✓	✓			
tcp.window_size_scalefactor					✓	✓	
udp.dstport		✓		✓			
udp.stream					✓		
ip.flags.df	✓						
tcp.option_len	✓						
tcp.dstport		✓					
tcp.hdr_len		✓					
tcp.window_size_value			✓				
tcp.analysis.acks_frame			✓				
tcp.flags.push			✓				
udp.srcport				✓			
ip.dsfield.dscp				✓			
dns.flags					✓		
dns.flags.response					✓		
dns.qry.name.len					✓		
tcp.options.timestamp.tsecr						✓	
Σ (features)	9	10	10	9	10	10	1

V. Discussion

SysID performance results are similar to the IoT Sentinel's results. The average classification performance for the cumulative of all the devices were 79% for IoT Sentinel and 82% for SysID. IoT Sentinel was able achieve these results using a sequence of 12 packets whereas SysID uses only a single packet for fingerprinting. IoT Sentinel benefits from investigating a sequence of packets whereas we benefit from investigating the header field content. Another possible bias that IoT Sentinel might have is the consideration of link-layer packets. In SysID, we ignored link-layer protocols since it is highly likely for these protocol contents to have information specific to devices in a network. Our aim was to generate a general model based system which can be applicable to different networks.

Overall, SysID has several general advantages and unique features. First, certain device fingerprinting tools depend on specific information extracted from network protocols. The downside of these approaches is that in case of a change of the behavior of the protocol or its fields, these approaches would not be able to perform fingerprinting correctly. Although we detect and use such distinguishing features from the protocols, SysID can re-process new data after protocol change to re-extract new distinguishing features without expert supervision.

Since we extract features directly from the common protocol header fields, the fingerprints that we extract are stable. Such fingerprints can be used when a different environment or even mobility of devices is considered. Depending on which protocol header fields are used, SysID can generate fingerprints that are either very general or very specific depending on the need.

Different from other approaches, we do not hand select *useful* features such as packet size, port number or IP header options. GA helps SysID to automatically detect the set of the most useful features in a given dataset. Although the dataset used in this study contained fewer than 50 packets for some devices, SysID was able to detect the unique behaviors of IoT devices and obtain very high classification performances.

In particular, SysID performs a single-packet fingerprinting of devices using any of the packets generated from the device. This is useful for both reducing the complexity of classification and also to reduce expectations from the targeted device in terms of both the number of packets and a sequence of packets to be generated.

VI. Conclusion and Future Work

We present SysID, a completely automated single-packet IoT device classifier using machine learning and GA. GA helps in determining relevant features in packet headers to both increase classification performance and reduce complexity. In most cases, we were able to use less than half of the features in packet headers to classify devices at a rate higher than when all the features were used. GA also helps to eliminate noisy features that negatively affect the classification performance. We also analyze multiple machine learning algorithms to determine the best classifiers to fingerprint devices at each layer of classification. Overall, SysID was able to classify devices at a minimum accuracy rate of 95%. In some cases, SysID was not able to identify devices from the same vendor as they had similar network behaviors.

In the future, we would like to analyze groups of packets rather than a single packet. This would possibly allow higher classification performance. We would also like to extend our analysis to optimize GA. As we observed some of the selected features were not utilized by machine learning algorithms, we believe SysID could have achieved higher performance.

Acknowledgement

This material is based upon work supported in part by the National Science Foundation under grant numbers CNS-1321164 and EPS-IIA-1301726.

References

[1] B. Li, J. Springer, G. Bebis, and M. H. Gunes, “A survey of network flow applications,” *Journal of Network and Computer Applications*, vol. 36, no. 2, pp. 567–581, 2013.

[2] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, “Ddos in the iot: Mirai and other botnets,” *Computer*, vol. 50, no. 7, pp. 80–84, 2017.

[3] B. Li, M. H. Gunes, G. Bebis, and J. Springer, “A supervised machine learning approach to classify host roles on line using sflow,” in *Proceedings of the first edition workshop on High performance and programmable networking*. ACM, 2013, pp. 53–60.

[4] A. Aksoy, S. Louis, and M. H. Gunes, “Operating system fingerprinting via automated network traffic analysis,” in *Evolutionary Computation (CEC), 2017 IEEE Congress on*. IEEE, 2017, pp. 2502–2509.

[5] A. Aksoy and M. H. Gunes, “Operating system classification performance of tcp/ip protocol headers,” *Local Computer Networks Workshops (LCN Workshops), 2016 IEEE 41st Conference on*, 2016.

[6] G. F. Lyon, *Nmap network scanning: The official Nmap project guide to network discovery and security scanning*. Insecure, 2009.

[7] J. Schwartzenberg, “Using machine learning techniques for advanced passive operating system fingerprinting,” Master’s thesis, University of Twente, 6 2010.

[8] K. Gao, C. Corbett, and R. Beyah, “A passive approach to wireless device fingerprinting,” in *2010 IEEE/IFIP International Conference on Dependable Systems&Networks (DSN)*. IEEE, 2010, pp. 383–392.

[9] J. François, H. Abdelnur, R. State, and O. Festor, “Ptf: Passive temporal fingerprinting,” in *12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops*. IEEE, 2011, pp. 289–296.

[10] S. V. Radhakrishnan, A. S. Uluagac, and R. Beyah, “Gtid: A technique for physical device and device type fingerprinting,” *IEEE Transactions on Dependable and Secure Computing*, vol. 12, no. 5, pp. 519–532, 2015.

[11] D. Formby, P. Srinivasan, A. Leonard, J. Rogers, and R. A. Beyah, “Who’s in control of your control system? device fingerprinting for cyber-physical systems.” in *NDSS*, 2016.

[12] T. Kohno, A. Broido, and K. C. Claffy, “Remote physical device fingerprinting,” *IEEE Transactions on Dependable and Secure Computing*, vol. 2, no. 2, pp. 93–108, 2005.

[13] L. C. C. Desmond, C. C. Yuan, T. C. Pheng, and R. S. Lee, “Identifying unique devices through wireless fingerprinting,” in *Proceedings of the first ACM conference on Wireless network security*. ACM, 2008, pp. 46–55.

[14] N. T. Nguyen, G. Zheng, Z. Han, and R. Zheng, “Device fingerprinting to enhance wireless security using nonparametric bayesian method,” in *INFOCOM, 2011 Proceedings IEEE*. IEEE, 2011, pp. 1404–1412.

[15] Q. Xu, R. Zheng, W. Saad, and Z. Han, “Device fingerprinting in wireless networks: Challenges and opportunities,” *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 94–104, 2016.

[16] M. Miettinen, S. Marchal, I. Hafeez, N. Asokan, A.-R. Sadeghi, and S. Tarkoma, “Iot sentinel: Automated device-type identification for security enforcement in iot,” in *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*. IEEE, 2017, pp. 2177–2184.

[17] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, “The weka data mining software: an update,” *ACM SIGKDD explorations newsletter*, vol. 11, no. 1, pp. 10–18, 2009.