FlexHash: IoT Device Identification with Hybrid Locality Sensitive Hashing

Jay Thom, Nathan Thom, Batyr Charyyev, Emily Hand, Shamik Sengupta
Department of Computer Science and Engineering, University of Nevada, Reno, USA
1664 N. Virginia street m/s 0171 Reno, NV 89557 775-784-6905
Email: [jthom, bcharyyev, emhand, ssengupta]@unr.edu, nathanthom@nevada.unr.edu

***Abstract*—In recent times we have witnessed the rise of the *Internet of Things*, a vast network of relatively simple connected devices ranging from household appliances and convenience items, to connected devices and sensors used in commercial, industrial, and healthcare applications. In many cases IoT can provide efficient and affordable solutions, but can also introduce new problems related to network security. Low-powered devices often lack sufficient computational capacity or necessary security features to provide adequate protection. In addition, many devices come ready to connect *out-of-the-box* with default credentials that are often left in place, potentially leaving devices and networks vulnerable to attack. As characteristics such as MAC or IP addresses can be easily spoofed, device identification and monitoring has become a difficult task but one that is essential for network security.**

The problem of identifying IoT devices by fingerprinting their network traffic has been studied, with various approaches emerging. While achieving good results, many solutions require complex feature extraction, considerable computational overhead, and extensive domain knowledge in both networking and machine learning to select relevant features and supporting ML algorithms. In addition, many current studies work to identify heterogeneous devices in an artificially sterile (lab) environment. To improve the process we apply a combination of locality sensitive hashing and machine learning to identify specific devices based on a generic view of their network traffic characteristics. We successfully identify both heterogeneous and homogeneous (identical) devices in an environment with both live network noise and noise generated from other (unknown) IoT devices. FlexHash is able to consume unprocessed network traffic in the form of .pcap files and produce feature vectors capable of highly accurate device identification and anomaly detection. To enhance the strength of this approach we develop our own n-gram based hashing method allowing for various parameters in the algorithm to be tuned, making it possible to accurately differentiate individual devices from among a field of identical peers as well as device genre and heterogeneous devices with a single packet of network traffic.

***Index Terms*—Internet of Things, IoT Device Identification, Network Traffic Fingerprinting, IoT Security, Similarity Hashing**

I. Introduction

In recent times the concept of an Internet of Things (IoT) has been growing with numerous devices such as cameras, low-powered sensors, wearable technologies, and a multitude of household, commercial, industrial, and medical devices and sensors inter-connected to one another and to the Internet. The current digital landscape is essentially surrounded by IoT devices which play increasingly significant roles in our lives and have access to our private networks and data. This is a trend which is likely to continue, with reports showing there are as many as 30 billion IoT devices connected worldwide [1] with continued rapid growth expected. Along with the convenience offered by these technologies, there are also increased security risks with many devices under-equipped to prevent compromise, leaving themselves and the networks to which they are connected vulnerable to attack.

A first step in providing security for these systems and the networks on which they reside is proper accounting and management; tracking known devices, discovering rogue devices, and identifying abnormal or anomalous device behavior. This, however, can prove to be a challenging task. Common device identifiers such as IP and MAC addresses are of little use as they can be easily spoofed, and devices can connect to networks surreptitiously through rogue WiFi hot spots, requiring the development of alternate methods. The topic of IoT device identification by analyzing generated network traffic has been a popular area of research recently with several proposed solutions found in the literature. Various approaches including statistical, rule-based, and machine learning techniques have been explored which collect and analyze network traffic either in real time or offline. Rule sets or models for machine learning classifiers are developed which have been able to identify specific devices and device types as well as differentiating *normal* traffic from *anomalous* traffic. This has also been helpful for highlighting when new or unknown devices connect, or when familiar devices begin behaving in unexpected ways; an indication of compromise. Many of these methods are not without their weaknesses, however, requiring specific traffic intervals be collected, that adequate quantities are collected, or that classifiers are able to operate within available computational constraints. Most require feature extraction and engineering which can introduce a high degree of overhead, or high degrees of required domain knowledge to choose the appropriate features for the given classifiers. In addition, many classifiers require models be updated and retrained frequently to continue providing good results [2] [3].

While good results have been achieved, previous methods for IoT network traffic fingerprinting often focus on heterogeneous devices in a laboratory environment devoid of the background noises present in live networks. Many studies have relied on data sets provided by seminal studies such

as IotSentinel [4], which are made up either of completely different (heterogeneous) devices or of different types of devices from the same manufacturer. In a realistic network setting, an administrator will likely need to track many devices different types, but also multiple devices of identical make and model (i.e. many identical web cameras) all producing very similar traffic. In addition, this traffic will be mixed in the presence of background traffic noise, possibly from other unknown or un-modeled IoT devices. In our previous work [5] we found that while our initial efforts were highly effective with heterogeneous devices they performed poorly against groups of identical (homogeneous) devices on the same network, leading us to this current work.

One of the more innovative ways of identifying IoT devices in a network based on network traffic is found in [6], wherein device fingerprints are generated using locality sensitive hashes. Locality sensitive hashing is useful in performing similarity searches, and has proven to be of fundamental importance in numerous fields, such as spam email, machine learning, data mining, and malware detection [7]. In a similarity search, "given a collection of objects D with some similarity measure s defined between them and a query object q, retrieve all objects from D that are most similar to q according to the similarity measure s" [8]. Put more simply, a hash can be generated that is very similar for similar inputs, with minor changes causing only slight differences in the resultant digest. This is in contrast to the more familiar cyryptographic hash, wherein the changing of a single bit in an input will cause a complete change in the output. For IoT identification, a model can be built from several samples of traffic generated by the same device, stored in a database, then compared to previously unseen traffic to find the highest degree of similarity to approximate a match. In the same sense, a threshold can be selected to identify devices of similar type, or genre. This approach provides a means for device identification without the need for the complexities of machine learning. To perform this type of identification, a flow of traffic data must be captured and analyzed. Unfortunately, it has been shown that as the sample size decreases, for instance from a 10-minute to a 5-minute traffic sample, the accuracy degrades considerably, requiring that a large enough sample be collected for reliable identification. In addition, database searches can introduce overhead making real-time identification difficult.

Our proposed method, FlexHash, is a novel system combining the data generalizing capability of locality sensitive hashing with the power of machine learning to achieve highly accurate device fingerprinting based on their network traffic and device behavior while requiring only a single packet of traffic. By combining these two approaches a traffic fingerprint can be created that is capable of identifying network devices with a very high accuracy in noisy environments, even with groups of identical devices. This approach avoids the complexity of feature selection and engineering by hashing an entire packet, then converting the resultant digest directly into a feature vector. Subtle differences in the typically repetitive traffic produced by IoT devices are captured by similarity hashing, and can then be modeled with the help of machine learning algorithms to produce a device fingerprint. The ability to achieve a high accuracy using only single packets for device identification will allow for system monitoring by periodically sampling traffic to detect device membership, device genre, the presence of unknown devices, and changes in device behavior indicating traffic anomalies and possible compromise.

Contributions of this work are as follows:

- We provide an extensive review of related IoT network traffic fingerprinting approaches found in the literature.
- We develop FlexHash, a novel locality sensitive hashing algorithm that enables adjustments to the hashing parameters (accumulator length, window size, and n-grams).
- We implement a network traffic fingerprinting method combining FlexHash with machine learning, and perform accurate IoT device identification requiring only a *single packet* of network traffic.
- We evaluate this system by classifying device genre and *identical devices* in the presence of identical peers while also including realistic *background noise*.
- We collect traffic data from three categories of 8 identical IoT devices, which we share with the research community at *github.com/UNR-IoT-Fingerprinting/FlexHash*.

The remainder of this paper is organized as follows: section II provides a review of related works, section III will describe our methodology, section IV will provide experimental results and an analysis of our method, and finally section V will present a conclusion and future work.

II. Related Works

Studies exist in the literature on the topic of IoT network traffic fingerprinting using a variety of techniques. Many studies utilize machine learning approaches wherein specific features are extracted and are used to build a model for device classification, typically focusing on a specific subset of device characteristics. These features are then paired with an algorithm or ensemble of algorithms which produce optimal test results. Still other studies utilize a variety of non-machine learning methods. In this section related works are divided into *non-machine learning (non-ML)* and *machine learning (ML)* sections, with each subsection therein presented by focused feature set or analytical approach, along with a brief description of the methodology used.

A. Non-ML Approaches

1) Temporal Features: In [9] Noguchi et. al. focus on communication features found in packet header information captured from IoT device network traffic. Devices are identified by comparing traffic from a target device with previous samples stored per device in a data base. Similarity between samples is measured by digitizing features then comparing their Euclidean distance. A feature *weight* is calculated, with greater weight given to the most unique features and a threshold for similarity is chosen to determine whether traffic was likely generated by

the same device. In addition, the system will monitor device state by noting the time change pattern of the number of features which can be extracted from signals generated by a given device. The authors note conditions that could potentially limit this process, including changes in where the device is installed, possible multiple and changing interfaces on a single device, updates to the device software, OS, or firmware, and changes to the network. Such conditions could make it difficult to track the current state of a device or to detect and identify anomalies. In the best case, this system is capable of detecting changes in devices state and can identifiy new devices that join the network.

Similarly, in [10] Mazhar and Shafiq seek to characterize IoT traffic in terms of temporal patterns, volume, and target endpoints, and also consider security and privacy concerns. Their system passively collects traffic from over 200 smart home networks as they connect to cloud and other third party services with the help of an un-named gateway management software company. In this way they seek to addresses the problem of identifying only traffic in a test bed environment by using IoT data collected in the wild. They discover that while smart home IoT ecosystems may appear fragmented, they are in reality largely centralized due to reliance on a few popular cloud and DNS services which are hard-coded in their firmware. They note that devices often exhibit specific behaviors based on their particular functionality such as video streaming, diurnal usage patterns, etc. making it possible to fingerprint their network activity. They also note device back-ends typically connect to a limited number of service providers making them more centralized than they appear, and also contributing to their vulnerability to attack based on non-encrypted communication.

2) Rule Based Behavioral Fingerprinting: Seeking to identify anomalous traffic, Gill, Lindskog, and Zavarshy [11] develop a baseline *normal* profile using traffic discovery and classification, then work to detect traffic anomalies based on the following features; overall traffic, transport layer protocol traffic, traffic by destination socket, packet size, and IP flow. Variations on statistical measurements of these five areas are used to formulate a profile that is considered to be normal, with variations that fall outside of these parameters identified as anomalous.

Feng, Wang, and Sun [12] seek to build annotations for device types, vendors, and product names by leveraging application layer response data from IoT devices and relevant websites to obtain product descriptions. Honeypots are also used as input for eliciting response data from offending IP addresses found there. These annotations are used to build a rule set for devices within a network to control and restrict access and communication to and from devices.

3) Identification with Similarity Hashing: In an innovative work Charyyev and Gunes [6] use locally sensitive hashes to determine similarity in network traffic flows, a useful technique in that feature extraction is not required. Hashes are generated and compared for similarity using the Nilsimsa hashing tool, achieving a high degree of precision and recall when compared against a set of devices used to generate traffic flow samples in the same network. The system is tested on traffic collected from 22 devices by the authors, as well as on other publicly available data sets. Variable lengths of traffic flows are tested to determine how much traffic is sufficient to generate a good fingerprint. We take this approach further by extracting each byte from generated hashes and apply them as features to be used with various machine learning approaches to improve performance.

B. ML-Based Approaches

1) Temporal Features: In [13] Aneja et. al. perform device fingerprinting based on packet inter-arrival times in combination with deep learning. A Raspberry-Pi is configured to sniff network packets, and graphs are plotted for arrival times to two Apple devices, an iPhone and an iPad. A Convolutional Neural Network (CNN) is used to classify devices based on the generated graphs, achieving a device identification accuracy of 86.7%.

2) Network Layer Features: Meidan et. al. employ supervised learning to train a multi-stage classifier in [14]. The first stage of the classifier works to categorize traffic as *IoT* or *non-IoT* traffic. A subsequent second stage further attempts to categorize devices by *class* to determine device type. Individual features are extracted from packets by analyzing distinct traffic flows which are determined by source and destination IP addresses and port numbers from the SYN to FIN flags. Flow data is then enriched with publicly available data sets such as *Alexa Rank* and *GeoIP*. Finally, data is separated into 3 sets for classification; two for training (single session and multi-session classifiers) and one for verification. This method achieves an accuracy of 99.28% in identifying a pre-defined set of heterogeneous devices.

3) Application Layer Features: In [1] Ullah and Mohmoud focus on the set of application layer services a device uses, such as ARP, SSL, LLC, EAPOL, HTTP, MDNS, and DNS to develop a static view of device behavior. A data set adapted from the Aposemat IoT-23 Pcap dataset is used for testing, and both packet headers and payload are considered. Five steps are followed to identify devices; monitoring, building of a sensor profile, analysis of results, device classification, and prevention & recovery. 3 *K-fold* cross-validation tests are used to measure feasibility and model over-fitting. The authors claim a device identification accuracy of 100%, although it appears there are only three heterogeneous devices included in the data set.

4) Initial Connection Phase Fingerprinting: Miettinen et. al. develop IoT SENTINEL [4] with the goal of restricting communications in an IoT network to prevent an adversary from connecting to vulnerable devices, or to use a compromised device to communicate with or ex-filtrate data from other vulnerable devices on the network. A fixed set of features are captured during a devices *initial connection phase*, which are used to train a two-fold clasification process. In the first phase, a binary classification is performed to determine whether or not the device in question is in fact an IoT device.

In the second phase, the first 12 unique vector packets are captured to create a fixed-length vector which is used to train a classifier. A series of fingerprints are constructed which are then mapped to device types. Additional rules can be applied to device types based on externally available information regarding their potential vulnerability. By using only features extracted from IP packet headers rather than payload information, the system is able to avoid issues with encrypted traffic. The method claims an overall average accuracy of 81.5% against a set of heterogeneous devices, but traffic must be captured during the initial connection phase.

5) Identification of Device Genre: In [15] Marchal et. al. seek to define policies for various classes of IoT devices based on device type rather than on specific device. By monitoring network traffic, *AuDI* autonomously fingerprints devices in any state of operation *after* the initial connection phase. Since the goal of *AuDI* is network management, device classes rather than individual devices are identified as abstract *device types*. Once fingerprints are captured, an unsupervised clustering algorithm is used for classification. Policies are then formulated for appropriate device limitations which are stored in a database. These are used for anomaly detection, network resource allocation, and identification and isolation of vulnerable devices. AuDI claims an accuracy of 98.2% in identifying device type (genre) using a data set collected from 33 heterogeneous off-the-shelf devices.

6) Flow-Based Feature Extraction: In [16] Salman et. al. develop a framework for device, traffic type, and anomaly detection by observing traffic flows and applying statistical analysis. For each flow of 16 consecutive packets 4 simple features are extracted; packet size, direction, timestamp, and transport protocol. To classify devices the authors utilize decision tree, random forest, recurrent neural networks, residual neural networks, and convolutional neural networks. Once a model is developed, traffic can be monitored in real time. The authors claim to achieve 94.5% accuracy for device-type identification, 93.5% accuracy for traffic-type classification, and 97% accuracy in detecting abnormal traffic.

Silvanathan, Gharakheili, and Sivaraman develop a set of one class clustering models in [17] to identify devices from a real-time flow level telemetry. The primary features used for model training are average packet size and average flow rate. Per-flow packet and byte counts are captured each minute, and attributes are computed at time granularity increments of 1, 2, 4, and 8 minute intervals. The authors use their own packet-level parsing tool, which takes in raw *.pcap* files as input, develops flow tables, and exports byte and packet counters of each flow. finally, a vector of attributes is generated each time interval which corresponds to each device. Clusters are identified using a K-Means algorithm, which are then used to identify device instances.

Using a hybrid deep learning approach, Bao, Hamdaoui, and Wong address the issue of white listed, new, and anomalous devices in IoT networks in In [18]. The method combines clustering with deep neural networks to enable the classification of both previously seen and unseen devices, and uses an auto-encoder technique to reduce the dimensionality of the resulting data set. An unsupervised data clustering algorithm called OPTICS is employed which is based on space density. The auto-encoder is a symmetrical artificial neural network for reconstructing a given input. Known devices are identified using a random forest classifier, based on an input vector with 297 features extracted from the network traffic.

Similarly, Hamad et. al. [19] address the problem of IoT device identification by assembling a series of packets from network traffic flows, and extract various features to create a fingerprint for devices. Supervised machine learning algorithms are then applied on these features to solve a multi-class classification problem. Once devices are identified, rules can be applied to restrict their privilege to communicate on the network. In this way, a white list of known devices can be applied, and abnormal or unwanted traffic can be identified and blocked. A vector of 67 features are selected, and algorithms such as Adaptive Boosting, Latent Dirichlet Allocation, K-Nearest Neighbor, Decision Tree, Naive Bayesian, Support Vector Machine, Random Forest with 100 estimators, and Gradient Boosting are applied. In addition, summary statistics on network features are calculated.

7) Image Analysis: In a unique approach, Kotak and Elovici address the issue of BYOD, employees connecting their own personal devices to an enterprise network [20]. In this scenario, a white list of permitted devices must be constructed to identify and assign rules to connected devices, and non-permitted devices must be identified. To accomplish this, the authors collect TCP traffic (UDP and other protocols are ignored in this study), which are processed to remove packet headers, and the resulting data is converted into a gray scale image with each pixel representing a hexadecimal value from the binary file. A single class classifier is applied to white listed devices, while a multi-class classifier is applied to non-white listed devices to identify anomalous traffic.

8) Single Packet Feature Extraction: For real-time monitoring, Aksoy and Gunes [21] develop a framework they call *SysID* which randomly selects single packets for analysis using a genetic algorithm (GA) to determine which features are relevant for classification. Features are then run on several different machine learning algorithms such as Decision Table, J48 Decision Trees, OneR, and PART for classification and identification. In this way, rules can be applied to limit communication from the gateway or firewall to the IoT device to provide device-appropriate security. SysID can perform single packet analysis on network traffic selecting features without the need for expert input, and achieves over 95% accuracy.

Using a similar approach, in [22] Chowdhury et. al. perform device identification through fingerprinting by extracting features from single TCP/IP packet headers. A feature vector is developed by generating a score for all available features based on variability, stability, and suitability of each bit. WEKA tool is then utilized to sample various ML algorithms for device classification. They test their work on two publicly available

data sets (UNSW and Iot_Sentinel), and achieve an accuracy of 97.78%.

9) Behavioral Fingerprinting: In [23] Bezawada et. al. ask the question *what is this device* and *is this device the one it claims to be*. To accomplish this, the authors perform device behavior fingerprinting by extracting features from its network traffic. They attempt to approximate the device type based on an analysis of a collection of the protocols used, and by observing command and response sequences elicited from a device via its smart phone app. Features extracted include both packet headers and payload-based features. Various machine learning classifiers are employed from the *scikit learn* suite to identify device types. The authors claim a device can be identified with as few as 5 packets from a single device.

Also interested in device behavior, in [24] Yousefnezhad, Malhi, and Framling analyze packet header information to capture statistical information, and combine this with sensor measurements to form a feature set that is used in classifying IoT devices. Their method is used in both normal and under-attack scenarios, and are able to identify devices with a high degree of accuracy. Seven behavioral profiles are developed that are used to develop rules for network security. Various machine learning algorithms are applied depending on the degree of linearity of the captured data.

10) Limited Feature Extraction/Engineering: To address the problem of feature selection, Fan et. al. [25] work on IoT device identification using semi-supervised learning. The main advantage of this approach is that rather than having to determine a large set of features for model training, a smaller set of labeled data is used along with a much larger body of unlabeled data, with labels inferred from the smaller set. To facilitate this, labels from the labeled set should adequately differentiate devices from one another. In addition, IoT devices should be differentiated from non-IoT devices that exist on the network to improve model accuracy. Features chosen include time intervals, traffic volume, protocol features, and transport layer security features. Transformed features are clustered per class to compensate for feature fluctuation. Convolutional Neural Networks are used for dimension reduction and multi-task learning is employed, helping the classifier to distinguish IoT devices from the rest of the traffic. The authors claim to achieve an average accuracy of 99.81%.

11) Textual Features and Data Mining: In [26] Ammar, Noirie, and Tixeuil use a *Bag of Words* approach to identify IoT devices on a network. A series of feature sets are collected both actively and passively (depending on the device) from packets containing service discovery protocols, DHCP fingerprinting, and user agent information found in the HTTP headers. This textual information is modeled as binary data using Bag of Words. Devices are represented by a feature vector of m words, and is set to 1 if a word is present in a device description, and 0 if it is not. Vectors are added to a database to construct an p X m matrix where columns represent words and rows represent devices. These are then used as device labels.

In a separate work, [27] Ammar, Noirie, and Tixeuil utilize a supervised learning classifier to differentiate devices based on features from network flow, as well as textual features from packet payloads. Devices are identified within a home network to determine their specific capabilities. This method seeks to identify a device as it joins the network and is in its connection setup phase. Features that are captured include maximum and minimum packet length, average packet inter-arrival time, number of packets in a given flow, protocols used in the flow, device manufacturer name, model, friendly name, XML description, mDNS information, device OS, and device model. Textual features are modeled as a bag-of-words, with each device description making up a feature vector.

12) Feature Cost Metrics: Finally, in [28] Desai et. al.focus on the cost of choosing and selecting appropriate features for applying machine learning algorithms to network traffic for device identification. Their framework selects features based on their utility for meeting the needs of specific algorithms. Once features are selected, popular ML algorithms are applied. A statistical method is also utilized to screen features for their relative value.

Non-ML approaches offer the advantage of simplicity, but often require larger traffic samples for comparison, and potentially large databases which must be searched to find matching device profiles. Machine learning solutions offers the advantage of highly accurate identification with smaller traffic flows and lend themselves to real-time monitoring which is more difficult when a database is required for comparisons, when larger flows are required, or when rules must be applied for identification. While effective, machine learning approaches rely on feature extraction, selection, and engineering, a process that can be computationally expensive and typically requires a high degree of domain knowledge both in machine learning and networking to select appropriate and relevant features for a given algorithm. FlexHash takes advantage of both approaches by combining a hybrid similarity hashing technique with machine learning, giving it the strengh and real-time monitoring capability using only a single packet of network traffic for highly accurate identification, but requires no feature extraction or engineering. In the following section the details of this system will be described.

III. Methodology

FlexHash relies on the principle of Locality Sensitive Hashing (LSH) to convert .pcap files into feature vectors. The resultant vectors are suitable for use with machine learning algorithms to produce IoT network traffic fingerprints. This section will provide a description of the proposed hashing method *FlexHash* and show how it is used in combination with machine learning algorithms to identify IoT devices. We will also provide details of the data set used for evaluation.

A. Locality Sensitive Hashing

Unlike cryptographic hashes which produce an entirely different output if even a small change is made in the input data, locality sensitive hashing produces similar outputs for similar inputs. If a small change is made in the input data, only a small change will result in the output. LSH has been

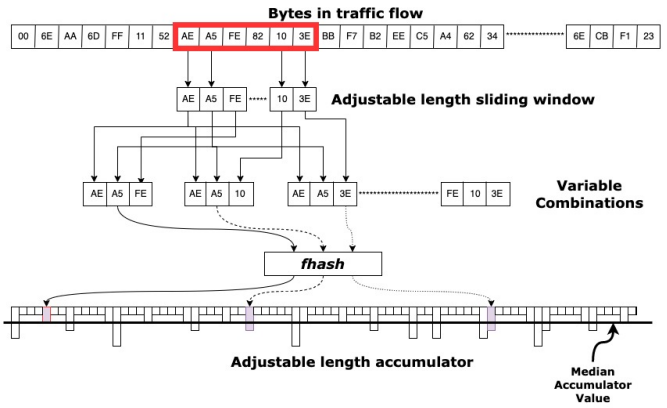


Fig. 1: FlexHash functionality.

utilized for a number of tasks including electronic identification [29], anomaly detection [30], malware classification [31], and spam email detection [32] [33]. Nilsimsa [32], ssdeep [34], sdhash [35], and tlsh [36] are the most commonly known n-gram based LSH approaches. In general, these hash functions operate by using a sliding window of a specific size that moves over the input stream one byte at a time. In each iteration, trigrams are generated from the current characters in a window which are then passed to a hash function to generate integer index values. Using these values, frequencies of the observed trigrams in the input are stored in an accumulator array and a hash digest is created based on these stored values. We propose a novel n-gram hash function *FlexHash* and demonstrate its effectiveness in identifying IoT devices.

The main objective in employing LSH is to reduce the dimensionality of a network packet by transforming the contained data from a high to a low-dimensional space so that the new representation retains some meaningful properties of the original including the essence of a devices recurring network behaviors. Secondly, we must have the ability to adjust the parameters of the LSH algorithm such that its functionality can be tuned to allow optimization of the output for individual devices. In our previous work we find that Nilsimsa works well for fingerprinting heterogeneous devices exhibiting clearly different network behavior, but is less effective when used to hash traffic for identical devices. For this reason, we develop a new n-gram based hashing method, namely FlexHash.

B. FlexHash: Hybrid LSH

Compared with other n-gram based approaches (i.e. *Nilsimsa*), FlexHash provides greater flexibility and improved results (Table IV) by allowing parameter tuning (sliding window, n-gram, and digest length) to be used with specific devices. Table I provides the tunable parameters along with the set of ranges chosen for each parameter for this study. Details of FlexHash's functionality are provided in Figure 1. FlexHash utilizes an adjustable size sliding window that reads the input one byte at a time. At each iteration, all combinations of bytes in the window are generated and are passed to the hash function *fhash*. This function (fhash) multiplies the cumulative integer value of each combination. It then adds a prime number to the result to facilitate greater hash distribution. The result is divided by the length of the accumulator and the remainder (modulo) becomes the current index. The value in that index of the accumulator, which is initialized at zero, is incremented by one. As a result, the accumulator holds frequencies of the hashes of each observed combination taken from an input. The final digest of the input will be the same length as the accumulator. Finally, for every position (address) in the accumulator, if the value at a given address exceeds the median value of all indexes then the value of that address is set to 1, otherwise it is set to 0 to create a digest. The window size and accumulator size are adjustable and can be tuned based on the problem and input characteristics. The n-gram (combination) size is also adjustable with the upper limit being the current window size. Pseudo code for this process is presented in Algorithm 2.

To perform optimal parameter selection for a particular device, we run pre-chosen combinations of parameters and test which set provides the best results. Optimal parameters vary based on the device type, making the adjustable property of FlexHash vital to improving accuracy and other performance metrics, especially when identifying identical devices. Optimal parameters for devices found in our evaluations are *1024, 6, and 2* for accumulator, window, and combination size for smart plug and light bulb, and *1024, 4, and 2* for web cameras. As a future work, an automated process to predict a range of optimal parameters for a given device type would be desirable.

C. fhash algorithm

In order to capture similarity and produce accurate fingerprints of device network activity .pcap files must be hashed

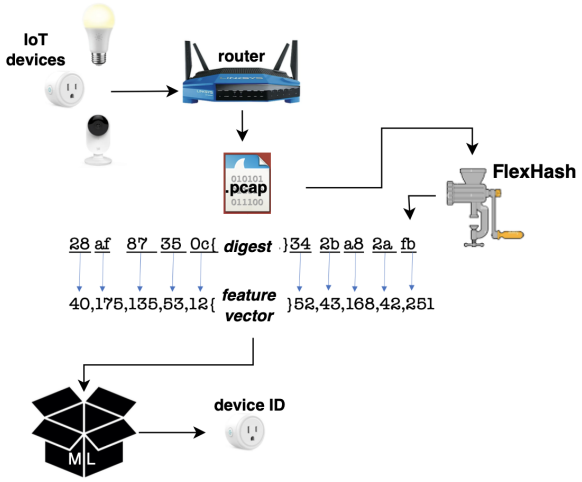


Fig. 2: The network traffic is continuously monitored by the device identification system. The traffic data is processed by FlexHash and converted into feature vectors. These feature vectors are given as input to the pre-trained ML model to identify the device that generated the traffic.

TABLE I: Tunable Parameters of FlexHash

Accumulator Size (in bits)	128, 256, 512, 1024
Window Size	4, 5, 6
Combination Size	[2: Window Size]

such that there is an even distribution of hash collisions across the length of the accumulator. We design our hashing algorithm *fhash* to meet his need. fhash receives a set of combinations and returns a hash value in the range (0, [accumulator length]-1) which represents an array address. Segments of a .pcap file that are sufficiently similar will output the same (or similar) addresses. This allows FlexHash to capture the essence of device behavior, especially as it relates to repetitive activities which are common in IoT devices. Pseudocode for the process of hashing .pcaps and producing appropriate list indexes is presented in Algorithm 1.

Algorithm 1 fhash

```
ngram list ← current ngrams
total ← 1
for each ngram do
    total ← total * ngram
end for
result ← total + 3
result ← result % accumulator length
return result
```

D. Hashing .pcap Files

As mentioned earlier, FlexHash avoids the complexity of feature selection and instead applies the entirety of a .pcap file in hashed form to produce a fingerprint. Earlier studies often work to capture network features from packet headers such as protocol, packet size, packet inter-arrival time, etc. to find distinct qualities with which to produce a recognizable uniqueness. What is missed in this approach is what lies below the surface, in the message body of the packet. Each packet produces traffic in which it identifies itself in various ways, either sharing its MAC address, current IP address, or other details that may specifically identify an individual device. Even if identifying features are removed from the packet headers to avoid bias in the machine learning algorithm (as they are in our study), they may still appear in the encapsulated message body, and can be captured by a hash regardless of encryption, etc. This is because the hash seeks to find not specific information, but similarity to other packets produced by the same device. An example of this can be seen in figure 4. In the figure we see data from the various layers of the packet; but as they are peeled away we see at their core the message body which holds data the device is sending out or receiving as it communicates with the network, other devices, or the server/website of the device manufacturer. We believe this contributes heavily to the effectiveness of our approach.

Algorithm 2 FlexHash

```
x ← window size
y ← ngram size
z ← accumulator size
procedure MAKEHASH(x, y, z)
    for .pcap in file do
        counter ← 0
        for bytes in .pcap do
            current ← window[counter:counter+x]
            ngramlist ← current
            procedure FHASH(ngramlist)
                return index
                accumulator[index += 1
            end procedure
        end for
    end for
    total ← 0
    while index < z do
        total ← total + accumulator[index]
        median ← total / z
        for accumulator[x]=0; x < z; x++ do
            if accumulator[index] ¡ median then
                accumulator[index] ← 0
            else if accumulator[index] ¿ median then
                accumulator[index] ← 1
            end if
        end for
    end while
    vector = []
    counter ← 0
    for bits in accumulator do
        current ← accumulator[counter:counter+8]
        current ← int(current)
        vector.append ← current
        counter ← counter + 8
    end for
end procedure
```

E. Identification of IoT Devices

The overview of the IoT device identification system is presented in Figure 2. Traffic data is captured from IoT devices in a network and hashes of the traffic data are generated with FlexHash. These hashes are converted to feature vectors by converting each byte value in the resultant digest to base-10 numerical values ranging from 0-255. For instance, byte value ”FB” would be converted to the integer value 251. These feature vectors are then passed to our device identification system to train the underlying machine-learning model. Once a model is trained the identification system is ready for deployment on a router or micro-controller to serve as a gateway to a network. The device identification system will continuously monitor traffic data by capturing periodic samples to identify known devices, new devices joining the network, and to identify changes indicating anomalous behavior. This

is done by generating the digest of randomly captured traffic data in the form of packets, converting them to numeric feature vectors, and passing them to a pre-trained machine learning classifier.

F. Machine Learning

As a machine learning framework we use Autogluon-Tabular (AGT) [37] from Autogluon version 1.0.0. AGT is an automatic machine learning framework designed specifically for tabular datasets, such as spreadsheets. AGT aims to simplify the application of machine learning techniques for practitioners and researchers. We elect to use this framework because it enables streamlining the use of various best practices and machine learning strategies that lead to superior performance. The components of AGT that make it an effective framework for our data are the use of diverse base model types and carefully designed ensembling strategies.

AGT makes use of various well-known methods as base models. Base models are the underlying learning strategies used by AGT for modeling the input data. Each of these models has the capacity to perform well independently, but differences in their behaviour lead to variance in the patterns recognized. Ensembling predictions from models with different perception of the data space leads to robust final predictions. Base models available are Random Forest, Extra Trees [38], CatBoost [39], LightGBM [40], XGBoost [41], Logistic Regression, K-Nearest Neighbors (KNN), and Neural Network.

We select three base models to use in our experiments. These are Extra Trees, LightGBM, and XGBoost. These models where selected based on processing time and performance in preliminary experiments. Extra Trees, an ensemble learning method with additional randomness, builds multiple decision trees for predictions, potentially enhancing robustness at the cost of increased computational expense. LightGBM, a gradient boosting framework, prioritizes speed and efficiency with a tree-based learning approach, particularly suitable for large datasets but potentially more sensitive to overfitting. XGBoost, an optimized gradient boosting library, achieves exceptional performance through regularized learning and parallel computing, necessitating careful hyperparameter tuning.

Ensembling the predictions of multiple models leads to a reduction in the variance of a machine learning system and improved prediction accuracy [42]. AGT provides the implementation for two important forms of ensembling: repeated k-fold bagging and multi-layer stacking [43], [44]. Multi-layer stack ensembling is a strategy in which the output predictions of a preliminary layer of models are used as features or inputs to a subsequent layer. AGT's implementation also utilizes a form of skip connection which concatenates the original model features onto the preliminary layer's predictions. This technique allows the subsequent layers of models to have an understanding of the original data space in addition to previous layers' predictions. Repeated k-fold bagging is a method which randomly splits the data into chunks and trains multiple models on different random chunks of the available data. When making a prediction, the input data is passed through each model and the outputs are voted on. Whichever output is most frequent is used as the output for the group.

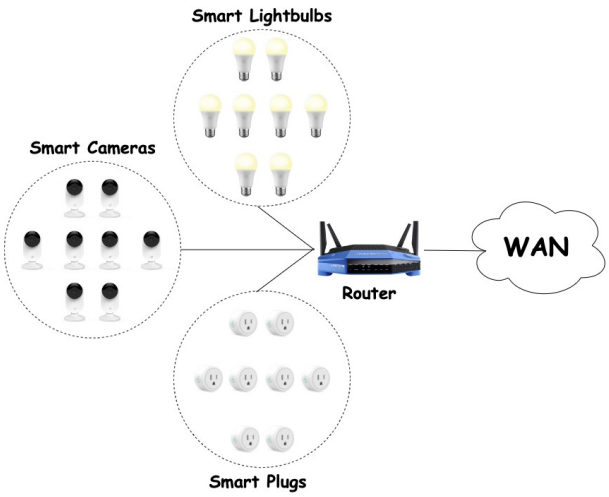


Fig. 3: Traffic data was collected from 3 categories of devices, representing simple to complex set of devices. Each set contains 8 identical devices. Data is collected for 24 hours.

The final model produced by AGT is a combination of 68 total models in 2 layers. All bags contain 8 copies of the base model trained on different portions of the available data. The first layer (models trained on just input data) contains 2 Extra Trees models, 3 bags of LightGBM models with varying hyperparameters, and 1 bag of XGBoost models. The second layer is identical to the first, except models are trained with the output predictions of the first layer models concatenated to the original input data. AGT utilizes a method called ensemble selection [45], which takes predictions from all available models into account and produces a single output.

G. Dataset

For evaluation we focus on three categories of devices; smart plugs (*Ghome Smart Plug*), smart light bulbs (*General Electric CYNC Full-Color Smart Bulb*), and web cameras (*YI 1080p Home Camera*). Each category of device contains 8 identical members for a total of 24 devices. Devices are connected to the network and allowed to complete their initial setup phase and then .pcap data is collected on a resting state for 24-hours. Data is then sanitized and header checksums are recalculated using *editcap*, replacing all MAC and IP addresses of the devices with 11:11:11:11:11:11 and 1.1.1.1 respectively to avoid bias introduced by unique addresses. Digests of each packet for each parameter combination are generated and converted to feature vectors as described above. For evaluation the data was split into 80% for training and validation, and 20% for testing. Results are measured in terms of precision (i. e., $TP/(TP+FP)$), recall (i. e., $TP/(TP+FN)$), f1-score (i. e., $2/(1/precision + 1/recall)$), and accuracy (i. e., $(TP+TN)/(TP+FP+TN+FN)$) where TP, TN, FP, and FN stand for true positive, true negative, false positive and false negative.

TABLE II: Average performance in identifying device genre in the presence of noise and without noise.

Device	Accuracy		F1 Score	
	Without Noise	With Noise	Without Noise	With Noise
Smart Plugs	99.98	99.89	99.97	99.90
Smart Lights	99.88	99.41	99.92	99.57
Smart Cameras	99.99	99.98	99.99	99.98

IV. Experimental Analysis and Results

In this section, we analyze the performance of our device identification system using FlexHash with the following experiments.

- To demonstrate the system's ability to operate in a realistic network setting we consider the effect of adding background noise to each data set. To achieve this we combine device data with two types of noise: *IoT noise* (network traffic of random IoT devices), and *network noise* (random traffic from a live network).
- We generalize all devices into three categories: *smart plug*, *smart bulb*, or *web camera*. A model is built and packets from each device are classified by *genre*.
- We determine the source device of a network packet from a pool of 8 of the same model devices. We train a model, then identify packets from that group as belonging to a distinct individual.
- We compare individual device results to those from another popular LSH method, namely *Nilsimsa*.
- Finally, to show the effect of parameter tuning we generate confusion matrices and UMAP [46] visualizations of the data with the best and worst parameter sets.

A. Identify devices in the presence of background noise

In a live network traffic noise is inevitable. Thus, it is crucial to explore system performance in a noisy environment. This demonstrates the ability to apply device identification techniques in a realistic network setting when monitoring for IoT device membership or when searching for unknown devices by category. To achieve realistic results, we introduce two types of background noise: *IoT noise* (random IoT traffic from [4]) and *network noise* (random network traffic from a large set of heterogeneous devices. We create hashes of the noise data and add it to device data set labeled as either *network-noise* or *iot-noise*. Results show noise has a minimal impact on performance demonstrating the system's ability to function in a live network environment. Results for experiments with and without background noise can be seen in tables II, III, and IV.

B. Identify devices by genre

The identification of devices by genre enables inferring the device type for traffic captured in real-time via a single packet. By monitoring a network and testing random packets we can predict the likelihood that a particular packet belongs to a genre, i.e. the packet is *probably* a camera, or *probably* a smart bulb, etc. To do this, all 24 devices are re-labeled as either smart bulb, smart plug, or web camera, and a multi-class classifier is built. The implication here is that in a network scenario, randomly captured packets identified as being generated from some IoT device can be further categorized as a specific device type. This is useful for investigating unknown or rogue devices on a network that could potentially compromise security.

TABLE III: Average performance in identifying identical devices from the same category in the presence of noise and without noise.

Device	Accuracy		F1 Score	
	Without Noise	With Noise	Without Noise	With Noise
Smart Plugs	85.79	85.02	85.77	84.91
Smart Bulbs	93.63	89.09	93.60	84.75
Web Cameras	97.89	98.61	97.78	98.55

Results for this experiment are presented in Table II. Results for identification by genre are above 99% for all devices both with and without background noise. We are also able to differentiate the noise type from known device genres, achieving an accuracy for *iot-noise* above 98% and for *network-noise* above 97%.

C. Identification of individual devices from identical peers

In this subsection, we work to identify an individual device from a group of identical peer devices. Instead of attempting to identify a heterogeneous set of unique devices, we address the more challenging task of identifying clusters of identical devices, a task under-explored in the literature. This type of identification is critical in tracking device behavior and membership as there are often multiple individuals of the same

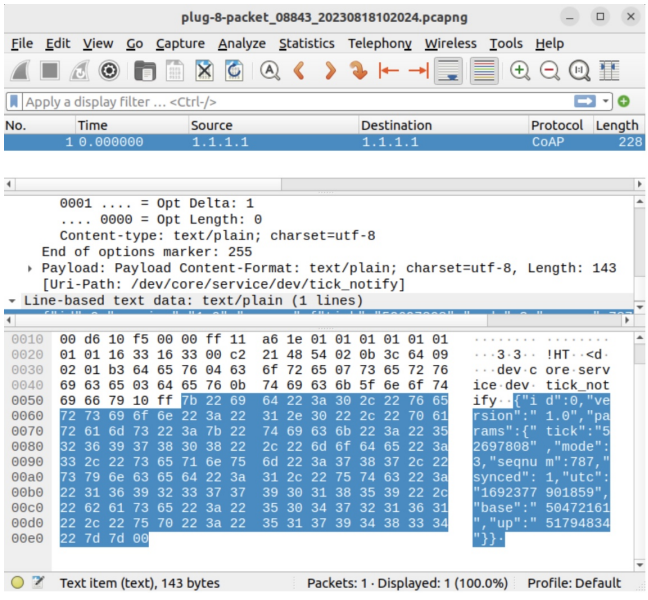


Fig. 4: Wireshark representation of a single *cleaned* packet (.pcap) including the hexadecimal representation of the packet contents which are processed by FlexHash.

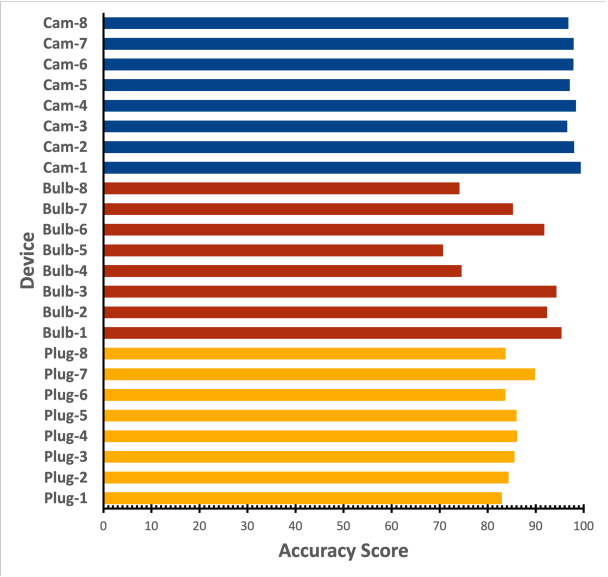


Fig. 5: Results for identification of individual devices from a group of identical peers (plugs, lights, and cameras).

device appearing in a network. Experiments are performed both with and without background noise and results compared.

Average results in terms of both accuracy and F1 score for this experiment are shown in Table III. We see that without background noise, web cameras achieve an accuracy above 97% on average while smart bulbs and smart plugs achieve results above 85% and 93% respectively. We note that in the case of smart plugs, some imbalance was found in the data samples for a 24 hour period, with three of the eight plugs generating considerably more traffic than the other five, possibly accounting for this change in performance. Interestingly, devices with greater complexity appear to perform better than simpler devices when identifying an individual from identical peers. We note here that FlexHash and our device identification system achieve accuracy results in this more difficult scenario that are either competitive or superior to results offered by other approaches in the literature while using only a single packet sample. Results per device are shown in Figure 5. When running the same experiment with background noise, web cameras, smart bulbs, and smart plugs achieve an average accuracy of 98%, 89%, and 85% respectively. With noise added, we only see a slight degradation of performance in the smart bulbs, with web cameras and smart plugs performing at nearly an equal accuracy.

TABLE IV: Comparison of average performance, FlexHash vs Nilsimsa in identifying identical devices from the same category.

	Accuracy		F1 Score	
Device	FlexHash	Nilsimsa	FlexHash	Nilsimsa
Smart Plugs	**85.79**	72.97	**85.77**	73.27
Smart Bulbs	**93.63**	78.04	**93.60**	80.48
Web Cameras	**97.89**	84.74	**97.78**	84.66

In Table IV we compare the performance of FlexHash with another n-gram based LSH method, *Nilsimsa*. In our previous study we observe Nilsimsa outperformed other similar methods such as ssdeep, sdhash, and tlsh. Thus, we can assume that better results with FlexHash over Nilsimsa will imply better results over other hashing techniques as well. Experiments are performed on network traffic data without background noise. FlexHash achieves an average accuracy of 97.74%, 93.63%, and 85.79% for web cameras, smart bulbs, and smart plugs respectively. On the same data, Nilsimsa achieves a lower performance of 84.74%, 78.04%, and 72.97%. This represents a percent increase of 13.00%, 15.59%, and 12.82%. We see here that in every case FlexHash achieves a significant increase in accuracy over Nilsimsa hashing, an indication of the effectiveness of FlexHash's tunable parameters.

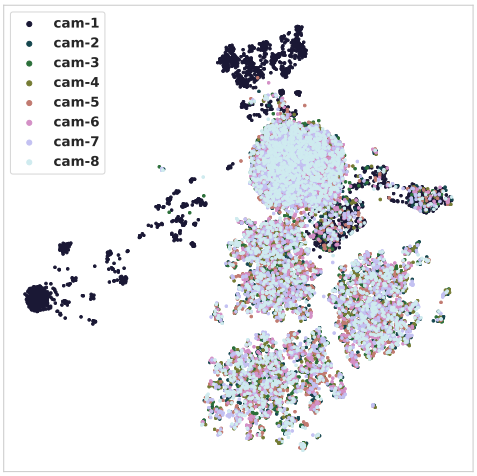
D. Visualization of FlexHash tunable parameters

In this section we demonstrate the strength of the FlexHash algorithm; hashing parameter tuning. As discussed above, our novel algorithm allows for the adjustment of variables that are pertinent to the hashing process (i.e. accumulator size, sliding window size, and combination size). The adjustment of these values allows FlexHash to capture more subtle aspects contained in the device network activity, producing hashes which are more effectively identified by the machine learning process when creating device traffic fingerprints.

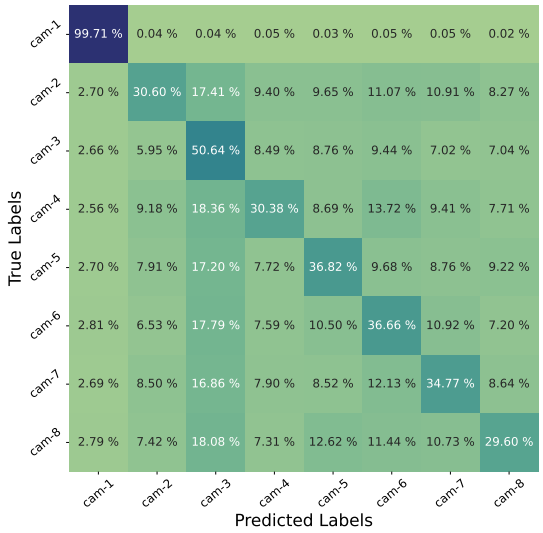
To aid the reader in understanding differences in parameter settings we utilize two types of visual aids; confusion matrices and UMAP representations [46]. A confusion matrix is a tabular representation of data that illustrates the performance of a model by displaying the counts of true positive, true negative, false positive, and false negative predictions. This enables assessment of the model's accuracy and error rates across classes, and helps determine misidentification. UMAP is a nonlinear dimensionality reduction technique used for visualizing high-dimensional data in lower dimensional space while preserving local and global structure. It focuses on finding a low-dimensional representation of data points while maintaining their high dimensional relationships to one another.

To demonstrate the affect of parameter tuning, we analyze the samples produced with the best and worst hashing parameters. Visualizations of these samples are found in Figure 6. Note that the UMAP plots are generated with only the hashed data and have no exposure to the machine learning model. We begin by inspecting Figure 6 (a) and (b) which reflect hashes generated from sub-optimal parameters. While cam-1 achieves high performance, the clusters of data points representing cameras 2-8 are are overlapping, thus the machine learning models struggle to accurately predict to which device a sample belongs.

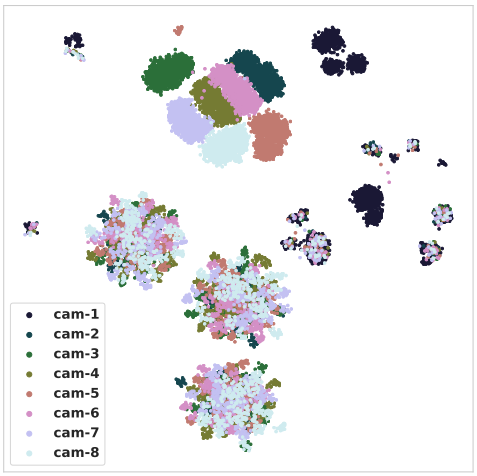
In Figure 6 (c) we see that by optimizing the parameter set used, FlexHash is able to hash devices into a representation which reliably separates data points originating from different devices into distinct spatial regions. The downstream effect of these parameters is shown in Figure 6 (d), with all devices



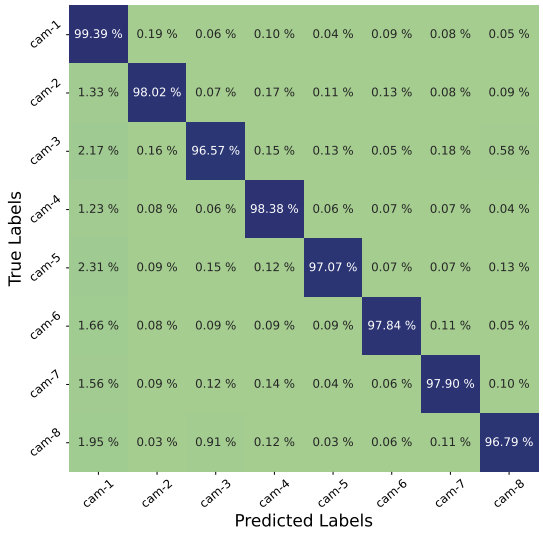
(a) UMAP representation of camera data hashed with accumulator length of 126, sliding window size of 6, and combination size of 6.



(b) Confusion matrix for the task of identical device identification on cameras hashed with accumulator length of 126, sliding window size of 6, and combination size of 6.



(c) UMAP representation of camera data hashed with accumulator length of 1024, sliding window size of 4, and combination size of 2.



(d) Confusion matrix for the task of identical device identification on cameras hashed with accumulator length of 1024, sliding window size of 4, and combination size of 2.

Fig. 6: Visual aids demonstrating the effect of tuning FlexHash's parameters. Subfigures (a)/(b) and (c)/(d) are pairs, with (a)/(c) visualizing the relationships of data points after hashing and (b)/(d) showing the performance of the best model trained.

being classified correctly 96% or more of the time. Locality sensitive hashing removes the need for feature engineering in device fingerprinting, while tun-able hash parameters enable adaptation of the hashing algorithm to various sets of IoT devices.

V. Conclusion

With the increased presence of IoT devices in both private and commercial networks, new methods must be developed to track device membership and behavior to prevent the compromise of both devices and network data. Rogue or infected devices can lead to compromise by unauthorized users, and can be weaponized to launch attacks such as malware installation

or distributed denial of service. As common identifiers such as MAC addresses are easily spoofed, methods such as network traffic fingerprinting are useful in providing management and security. While various methods have been proposed, many require the collection of specific traffic samples depending on size, device state, ect. to create rule sets, and samples must be compared to data bases to match pre-defined fingerprints for identification. Others utilize machine learning approaches, but require feature extraction and engineering, a process which can introduce considerable overhead, including the need for extensive domain knowledge in both machine learning and networking to develop feature vectors suitable for the chosen algorithms. In this paper we introduce FlexHash, a hybrid locality sensitive hashing approach that can convert unprocessed .pcap files into feature vectors without the need for feature extraction. FlexHash achieves a high degree of accuracy in network traffic fingerprinting and can successfully identify both heterogeneous and homogeneous devices as well as device genre in real time by identifying single packets. In addition, background noise is included to demonstrate this system's ability to function in a live network setting to assist with network monitoring and anomaly detection.

ACKNOWLEDGMENT

This material is based upon work supported by the National Science Foundation under award number 2019164. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] I. Ullah and Q. Mahmoud, "Network traffic flow based machine learning technique for iot device identification," in *2021 IEEE SysCon*. IEEE, 2021, pp. 1–8.

[2] R. Kolcun, D. Popescu, V. Safronov, P. Yadav, A. Mandalari, Y. Xie, R. Mortier, and H. Haddadi, "The case for retraining of ml models for iot device identification at the edge," *arXiv preprint:2011.08605*, 2020.

[3] R. Kolcun, D. Popescu, V. Safronov, P. Yadav, A. Mandalari, R. Mortier, and H. Haddadi, "Revisiting iot device identification," *arXiv preprint arXiv:2107.07818*, 2021.

[4] M. Miettinen, S. Marchal, I. Hafeez, N. Asokan, A. Sadeghi, and S. Tarkoma, "Iot sentinel: Automated device-type identification for security enforcement in iot," in *2017 IEEE ICDCS*, 2017, p. 2177:2184.

[5] J. Thom, N. Thom, S. Sengupta, and E. Hand, "Smart recon: Network traffic fingerprinting for iot device identification," in *2022 IEEE CCWC*. IEEE, 2022, pp. 0072–0079.

[6] B. Charyyev and M. H. Gunes, "Iot traffic flow identification using locality sensitive hashes," in *IEEE ICC*, 2020, pp. 1–6.

[7] N.Naik, P.Jenkins, N.Savage, L.Yang, T.Boongoen, and N.Iam-On, "Fuzzy-import hashing: A static analysis technique for malware detection," *Forensic Sci. Int.: Digital Investigation*, vol. 37, p. 301139, 2021.

[8] V. Satuluri and S. Parthasarathy, "Bayesian locality sensitive hashing for fast similarity search," *arXiv preprint arXiv:1110.1328*, 2011.

[9] H. Noguchi, T. Demizu, N. Hoshikawa, M. Kataoka, and Y. Yamato, "Autonomous device identification architecture for internet of things," in *2018 IEEE WF-IoT*. IEEE, 2018, pp. 407–411.

[10] M. Mazhar and Z. Shafiq, "Characterizing smart home iot traffic in the wild," in *2020 IEEE/ACM IoTDI*. IEEE, 2020, pp. 203–215.

[11] M. Gill, D. Lindskog, and P. Zavarsky, "Profiling network traffic behavior for the purpose of anomaly-based intrusion detection," in *2018 IEEE TrustCom/BigDataSE*. IEEE, 2018, pp. 885–890.

[12] X. Feng, Q. Li, H. Wang, and L. Sun, "Acquisitional rule-based engine for discovering internet-of-things devices," in *2018 {USENIX}*, 2018, pp. 327–341.

[13] S. Aneja, N. Aneja, and M. Islam, "Iot device fingerprint using deep learning," in *2018 IEEE IOTAIS*. IEEE, 2018, pp. 174–179.

[14] Y. Meidan, M. Bohadana, A. Shabtai, J. Guarnizo, M. Ochoa, N. Tippenhauer, and Y. Elovici, "Profiliot: A machine learning approach for iot device identification based on network traffic analysis," in *Proceedings of the symposium on applied computing*, 2017, pp. 506–509.

[15] S. Marchal, M. Miettinen, T. Nguyenand, A. Sadeghi, Ahmad-Reza, and N. Asokan, "Audi: Toward autonomous iot device-type identification using periodic communication," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1402–1412, 2019.

[16] O. Salman, I. H. Elhajj, A. Chehab, and A. Kayssi, "A machine learning based framework for iot device identification and abnormal traffic detection," *Transactions on Emerging Telecommunications Technologies*, p. e3743, 2019.

[17] A. Sivanathan, H. Gharakheili, and V. Sivaraman, "Inferring iot device types from network behavior using unsupervised clustering," in *2019 IEEE LCN*. IEEE, 2019, pp. 230–233.

[18] J. Bao, B. Hamdaoui, and W. Wong, "Iot device type identification using hybrid deep learning approach for increased iot security," in *2020 IEEE IWCMC*. IEEE, 2020, pp. 565–570.

[19] S. Hamad, W. Zhang, Q. Sheng, and S. Nepal, "Iot device identification via network-flow based fingerprinting and learning," in *2019 18th IEEE TrustCom/BigDataSE*. IEEE, 2019, pp. 103–111.

[20] J. Kotak and Y. Elovici, "Iot device identification using deep learning," in *Computational Intelligence in Security for Information Systems Conference*. Springer, 2019, pp. 76–86.

[21] A. Aksoy and M. Gunes, "Automated iot device identification using network traffic," in *2019 IEEE ICC*, 2019, pp. 1–7.

[22] R. Chowdhury, S. Aneja, N. Aneja, and E. Abas, "Network traffic analysis based iot device identification," in *Proceedings of the 2020 the 4th International Conference on Big Data and IoT*, 2020, pp. 79–89.

[23] B. Bezawada, M. Bachani, J. Peterson, H. Shirazi, I. Ray, and I. Ray, "Iotsense: Behavioral fingerprinting of iot devices," *arXiv preprint arXiv:1804.03852*, 2018.

[24] N. Yousefnezhad, A. Malhi, and K. Främling, "Automated iot device identification based on full packet information using real-time network traffic," *Sensors*, vol. 21, no. 8, p. 2660, 2021.

[25] L. Fan, S. Zhang, Y. Wu, Z. Wang, C. Duan, J. Li, and J. Yang, "An iot device identification method based on semi-supervised learning," in *2020 IEEE CNSM*. IEEE, 2020, pp. 1–7.

[26] N. Ammar, L. Noirie, and S. Tixeuil, "Network-protocol-based iot device identification," in *2019 IEEE FMEC*. IEEE, 2019, pp. 204–209.

[27] ——, "Autonomous identification of iot device types based on a supervised classification," in *2020 IEEE ICC*. IEEE, 2020, pp. 1–6.

[28] B. Desai, D. Divakaran, I. Nevat, G. Peter, and M. Gurusamy, "A feature-ranking framework for iot device classification," in *2019 IEEE COMSNETS*. IEEE, 2019, pp. 64–71.

[29] W. P. Filho, C. Ribeiro, and T. Zefferer, "An ontology-based interoperability solution for electronic-identity systems," in *2016 IEEE SCC*. IEEE, 2016, pp. 17–24.

[30] M. Pirker, P. Kochberger, and S. Schwandter, "Behavioural comparison of systems for anomaly detection," in *Proceedings of the 13th International Conference on Availability, Reliability and Security*, 2018, pp. 1–10.

[31] H. Kim, H. Song, J. Seo, and H. Kim, "Andro-simnet: Android malware family classification using social network analysis," in *2018 IEEE PST*. IEEE, 2018, pp. 1–8.

[32] E.Damiani, S.Vimercati, S.Paraboschi, and P.Samarati, "An open digest-based technique for spam detection." *ISCA PDCS*, vol. 2004, p. 559:564, 2004.

[33] M. Marsono, "Packet-level open-digest fingerprinting for spam detection on middleboxes," *International Journal of Network Management*, vol. 22, no. 1, pp. 12–26, 2012.

[34] N.Sarantinos, C.Benzaid, O.Arabiat, and A.Al-Nemrat, "Forensic malware analysis: The value of fuzzy hashing algorithms in identifying similarities," in *Trust/BigDataSE/ISPA*. IEEE, 2016, pp. 1782–1787.

[35] F. Breitinger and H. Baier, "Properties of a similarity preserving hash function and their realization in sdhash," in *2012 Information Security for South Africa*. IEEE, 2012, pp. 1–8.

[36] J. Oliver, C. Cheng, and Y. Chen, "Tlsh–a locality sensitive hash," in *2013 IEEE CTC*. IEEE, 2013, pp. 7–13.

[37] N. Erickson, J. Mueller, A. Shirkov, H. Zhang, P. Larroy, M. Li, and A. Smola, "Autogluon-tabular: Robust and accurate automl for structured data," *arXiv preprint arXiv:2003.06505*, 2020.

[38] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Machine Learning*, vol. 63, no. 1, p. 3–42, 2006.

[39] A. V. Dorogush, V. Ershov, and A. Gulin, "Catboost: gradient boosting with categorical features support," *CoRR*, vol. abs/1810.11363, 2018. [Online]. Available: http://arxiv.org/abs/1810.11363

[40] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017, p. 3149–3157.

[41] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 785–794. [Online]. Available: https://doi.org/10.1145/2939672.2939785

[42] T. G. Dietterich, "Ensemble methods in machine learning," in *Multiple Classifier Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 1–15.

[43] P. W. M. BAMBANG PARMANTO and H. R. DOYLE, "Reducing variance of committee prediction with resampling techniques," *Connection Science*, vol. 8, no. 3-4, pp. 405–426, 1996. [Online]. Available: https://doi.org/10.1080/095400996116848

[44] K. M. Ting and I. H. Witten, "Stacking bagged and dagged models," in *Proceedings of the Fourteenth International Conference on Machine Learning*, ser. ICML '97. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1997, p. 367–375.

[45] R. Caruana, A. Niculescu-Mizil, G. Crew, and A. Ksikes, "Ensemble selection from libraries of models," in *Proceedings of the Twenty-First International Conference on Machine Learning*, ser. ICML '04. New York, NY, USA: Association for Computing Machinery, 2004, p. 18. [Online]. Available: https://doi.org/10.1145/1015330.1015432

[46] L. McInnes, J. Healy, and J. Melville, "Umap: Uniform manifold approximation and projection for dimension reduction," 2020.