A Small World–Privacy Preserving IoT Device-Type Fingerprinting with Small Datasets

Maxwel Bar-on[1], Bruhadeshwar Bezawada[2(✉)], Indrakshi Ray[1], and Indrajit Ray[1]

[1] Computer Science and Engineering Department, Colorado State University, Fort Collins, CO 80526, USA
{Maxwel.Bar-on,Indrakshi.Ray,Indrajit.Ray}@colostate.edu
[2] Mathematics and Computer Science Department, Southern Arkansas University, Magnolia, AR 71753, USA
bbezawada@saumag.edu

Abstract. Internet-of-Things (IoT) device-type fingerprinting is the process of identification of the specific type of an IoT device based on its characteristics, such as network behavior. Such fingerprinting can be used to detect anomalous behavior of the device, or even predict its behavior should it get compromised. The typical approach to *fingerprint* an IoT device-type is by collecting a significant number of short network trace samples from these devices when it performs various activities and use machine learning on these samples to construct the fingerprint. There are several challenges to this approach. The first challenge is identifying the exact set of packets that correspond to the observed device-type behavior when it is performing some activity. The second challenge is that a single organization may not have enough data corresponding to all possible activities of the IoT device. We propose techniques to overcome the above mentioned challenges. First, to enhance device-type fingerprinting from small data sets, we designed a sliding-window based packet analysis behavioral model that provides improved data coverage associated with the activities of the tasks. Second, to get a model of the network behavior for the different activities of IoT devices deployed at various organizations, we use distributed deep-learning model so as to protect the privacy and confidentiality of the data. Finally, we alleviate the issue of data shortage by supplementing the training data with synthetic data generated using an Adversarial Autoencoder (AAE) neural network. We evaluated our approach using three different sets of experiments using a small set of representative devices. We estimate the best sliding window size for modeling device behavior by comparing the distributed learning performance over a range of window sizes. For our distributed approach, we achieve fingerprinting accuracy in the range of 94–99%, which is an improvement over the centralized approach for the same data sets and experiments. We demonstrate accuracy of 97%, on-par with state-of-the-art fingerprinting approaches, when using synthetic training data generated by our AAE. We note that, this is the first such method of fingerprinting device-types in a collaborative privacy preserving manner while alleviating small data sets.

M. Mosbah et al. (Eds.): FPS 2023, LNCS 14551, pp. 104–122, 2024.
https://doi.org/10.1007/978-3-031-57537-2_7

Keywords: Device-type Fingerprinting · Internet-of-Things · Privacy Preserving · Machine Learning · Collaborative Learning

1 Introduction

1.1 Motivation

Internet-of-Things (IoT) now has a default presence in home and organizational networks where they provide a wide range of services. A majority of IoT devices are plug-and-play, and any user can connect an IoT device to an existing network. However, an insecure or incorrectly configured IoT device is a significant security risk as these devices have proven [1] to be easy targets for attackers. Therefore, a network administrator needs to correctly identify the IoT device-types and enforce the necessary security controls to prevent and detect attacks.

The traditional device-type identification approach is to create a digital *fingerprint* from the network data generated by an IoT device and applying machine or deep learning techniques. Therein lies the major shortcoming of this approach, for a given timing window of packet capture, some devices generate a small amount of data while others generate larger volume of data making generalized analysis difficult. Therefore, a critical challenge in IoT device-type fingerprinting is to generate an accurate fingerprint even with a limited amount of data.

1.2 System Model and Assumptions

Our system model consists of a network of IoT devices whose network traffic can be captured by a network analyst. We assume the capability of capturing all kinds of traffic such as device-to-device, device to the Internet and from the Internet to a device. We do not make any assumptions on nature of the payload, which may be encrypted, compressed, binary or plain-text.

1.3 Problem Statement

The problem of device-type fingerprinting is to identify the device-type of an unknown IoT device based on the network traffic observed from the device. Let, $T = \{t_1, t_2, \cdots, t_n\}$ be the set of traffic traces collected from a known device-type D where each trace is a group of one or more network packets of fixed size and indicative of different behavioral features of the device. The goal is to build a machine learning model $M(T)$ that will output $True$ when tested on a sample trace T of the same device-type D, and output $False$ when tested on a sample trace of some other device-type. Ideally, the selected traffic-traces should encompass all possible behaviors of the device to enable a machine learning model to accurately identify the device regardless of the sample of traffic-trace being tested. However, it may not be possible or practical for one single observer to collect all possible network traces of a given device-type. Therefore, we assume that several entities are independently collecting sufficient number of traces from the device-type.

Now, the problem statement is as follows: to build a machine-learning model for fingerprinting IoT device-types using traces collected by diverse entities. Let $(D, P, T) = \{(P_1, T_1), (P_2, T_2), \cdots, (P_n, T_n)\}$ denote the collection of traces $T = \{T_1, T_2, \cdots, T_n\}$ of device-type D by a collection of parties $P = \{P_1, P_2, \cdots, P_n\}$ where P_i collects the packets in T_i and does not share with any other party. From these traces, the parties collaborate to build a single machine learning model $M(T)$ for accurate IoT device-type fingerprinting.

1.4 Limitations of Prior Art

Prior solutions [2–14] have one or both of the following limitations:

- *Identifying IoT Device-type Behavior.* For a network analyst, it is difficult to accurately identify the boundaries of the behavior of the IoT device-type as this requires a significant amount of data that encompasses the entire set of behaviors of the device. In prior art, analysts spent considerable time with the set of devices at their disposal to extract such behaviors, but it may not be practically possible for an independent analyst to cover all behaviors of the device.
- *Unknown Devices and Privacy of Data.* In prior art, if a new device is introduced into the network, the machine learning models need to be trained again, which is a significant effort and overhead. To avoid this, one may try to obtain data of unknown devices from other networks, but due to the privacy concerns this is difficult in practice.

In our work, we attempt to address the identified shortcomings of prior art and provide an accurate device-type fingerprinting model.

1.5 Overview of Proposed Approach

First, to solve the problem of small data sets, we make the best possible use of the collected device data with help of a sliding window packet analysis method. Further, we employ AAEs to generate synthetic data to improve the fingerprinting performance. We chose the AAE for our purpose, as opposed to a Generative Adversarial Networks (GANs), due to the limited volume of available data and the nature of our dataset. AAEs are better at handling heterogeneous data and require less training samples to produce high-quality synthetic data. In our experiments with synthetic data, 10% of samples in the training sets were generated by our AAE.

Second, to learn about unknown behaviors of devices and also, possibly unknown devices, we use a collaborative machine learning model where multiple parties train their local learning models using privately collected data and share the trained parameters using an aggregation algorithm. The end result is that the local machine learning models of individual parties are trained on the combined data of all the parties without any exchange of the actual data. The major advantage of the distributed deep learning model is that an individual party will be able to fingerprint unknown device-types without having trained on that device-type data.

Finally, we performed three major sets of experiments to validate our approach. In the first set of experiments, we compared the performance of the distributed learning model over a range of window sizes to estimate the window size which is optimal for modeling device behavior. We found that a window size of 10 packets gives the best performance with an average accuracy of 97%. In the second set of experiments, we tested the sliding window protocol using centralized and distributed learning models. Our results show that the distributed learning model performs on par and better when compared with the centralized learning models with an accuracy range of 94–99% in testing. In the third set of experiments, we tested the distributed learning model using real and mixed data sets where the mixed data sets contain 10% of synthetic data from the AAE. We found that the synthetic data did not degrade the model performance and achieved a similar range of 94–99% accuracy.

1.6 Key Contributions

First contribution is that, we describe an approach to generate machine learning models in a *privacy preserving* manner from small data sets. Second contribution, is the proof-of-concept usage of synthetic data for generating fingerprints of IoT devices, which are of similar quality to the fingerprints generated from real data. Third contribution is a comprehensive experimental validation of our fingerprinting approach. Here we show that our approach achieves a minimum-maximum true-positive rate (TPR) for fingerprinting IoT devices of 92%–99%, respectively, with a mean TPR of 95%, which is the performance of state-of-the-art centralized solutions.

Organization. In Sect. 2, we give an overview of related literature on IoT device-type identification and fingerprinting work. In Sect. 3, we give the detailed outline of our IoT device-type behavioral model using the sliding window approach, the feature engineering for accurate device-type fingerprinting and the privacy preserving deep learning model for collaborative fingerprinting. In Sect. 4, we describe the adversarial auto-encoder design for generating synthetic data. In Sect. 5, we describe our experimental evaluation and show the results of our fingerprinting approach. We summarize our work in Sect. 6 and outline important future challenges.

2 Related Work

2.1 IoT Device-Type Fingerprinting

Device-type fingerprinting has received considerable attention from the research community. General device fingerprinting [15–17] explore several techniques ranging from packet header features to physical features such as clock-skews. Wireless device fingerprinting techniques have been discussed in [10–14]. These works explore device-type identification by exploiting the implementation differences of a common protocol such session initiation protocol (SIP), across similar devices. Physical layer based device fingerprinting has received considerable

attention [18–22] where the focus is on analyzing the physical aspects of devices to fingerprint them. All these prior works focus on general wireless devices and their applicability to IoT devices is an open question.

Vladimir *et al.* [18] developed a radiometric approach based on imperfections in analog components for fingerprinting network interface cards (NICs). Such variations result in imperfect emissions when compared with the theoretical emissions and manifest in the modulation of the transmitted signals of the device. However, this work relies on the availability of link-layer frames from the given device, which may not be possible for IoT devices as they are spread over an area and are interconnected via different switches and routers.

François *et al.* [12,23] describe a protocol grammar based approach for fingerprinting. They characterize a device based on the set of messages transmitted. A message is represented using the protocol grammar syntax. To classify a given device, the messages emitted by the device are compared with syntactic trees of the stored fingerprints and depending on a similarity metric, the device label is assigned. However, this approach is again specific to protocols that are well known and whose grammar rules are available.

Gao *et al.* [24] develop a wavelet analysis technique to fingerprint wireless access points based on frame inter-arrival time deltas. The approach does not apply to IoT devices as these devices are usually end-points and do not forward data to other devices.

Radhakrishnan *et al.* [20] described a technique for device-type identification on general purpose devices like smartphones, laptops and tablet PCs using the inter-arrival times of different packets to extract features specific to a particular application, such as Skype. However, most IoT devices do not generate much traffic and obtaining such statistics will involve considerable time and effort.

Franklin *et al.* [10] describe a passive fingerprinting method for identifying the different types of 802.11 wireless device driver implementations on clients. The authors explore the statistical relationship of the active channel scanning strategy in a particular device driver implementation. This technique is useful for identifying the type of device driver implementation but not for the type of device.

In the IoT fingerprinting problem space, IoTSentinel by Miettinen *et al.* [25] and IoTScanner by Siby *et al.* [26] are the currently known solution frameworks. Miettinen *et al.* [25] describes IoTSentinel, a framework for device fingerprinting and securing IoT networks. It focuses on device-type identification at the time of device registration into a network. This approach uses packet header based features to identify a particular device-type and applies machine learning models to perform the fingerprinting. One shortcoming of this work is that it is susceptible to packet header spoofing. Our approach provides better accuracy and stronger security. IoTSentinel reports a mean identification rate of 50–100%, whereas our approach reports a mean identification rate of 93–99%.

Siby *et al.* describe IoTScanner [26], an architecture that passively observes network traffic at the link layer, and analyzes this traffic using frame header information during specific observation time windows. This work is more concerned with discerning the distinct devices and their presence based on the traffic

patterns observed during the traffic capture time window. A shortcoming of this approach is that two identical device-types could be classified as two different device-types due to the variations in traffic generated during the traffic capture time window. This approach is useful for network mapping at a high level, but performing this analysis periodically can be cumbersome. In [27], Bezawada *et al.* presented a centralized machine learning based model for fingerprinting IoT devices. They experimented with 23 different device-types and showed that the fingerprinting model was 99% accurate for identifying a majority of the device-types. We select this model for this work as it has a small number of features.

2.2 Privacy Preserving Deep Learning For Fingerprinting

The collaborative privacy preserving deep learning approach was described by Shokri *et al.* [28] and explored in depth by others [29–36]. The process considers a group of N-participants in the training process and a global server that maintains the final machine learning model and serves as a point of dissemination for the necessary information. During local training, each participant trains the neural network on its own private data using an optimization algorithm, such as Stochastic Gradient Descent (SGD). Once the training of the model begins, each client maintains a list of local parameters, *i.e.*, weight-gradients and bias-gradients. At the end of each training epoch, each participant selects exactly Θ_u most significant values from each layer, which contribute more towards the gradient descent, and shares them with the global parameter server.

To date, there is one recently published paper that focuses on federated learning for IoT (Fl4IoT) [37]. The work focuses on identifying unknown devices in the network. The first step of their approach is to use a clustering method to generate vector fingerprints, *i.e.*, short representation of a device, using an unsupervised clustering approach. Then, using supervised learning, they use these fingerprints to identify the presence of unauthorized devices in the network. Their work has similarities and dissimilarities with our work. The similarity is that both works attempt to fingerprint devices for the purposes of security. The dissimilarity is that their work focuses on identifying unauthorized devices based on a set of fingerprints identified in the given network whereas our work focuses on generalizing the fingerprint representation and can be used in unknown networks where we have not collected any data from the IoT devices. Importantly, our work focuses on using small data sets and a small number of features, making it simple to implement and deploy.

3 Device-Type Behavioral Model

In this section, we describe our behavioral model for fingerprinting IoT devices and the pertinent features.

3.1 Sliding Window Analysis for Behavior Coverage

We focus on designing the fingerprint of an IoT device-type based on the set of observable network behaviors of the device as we intend that the fingerprints are used by the network administrator for enforcing the necessary security policies. IoT devices typically provide specific services such as video camera monitoring the premises and sending the data feed to a network operator. A specific feature and/or service of an IoT device-type is obtained by sending a network command to the device. Therefore, a specific network behavior, b_i of an IoT device may be quantified by capturing the network packet trace t_i generated during such an interaction with the IoT device and extracting useful features. However, the network analyst performing the fingerprinting might not be able to identify the exact boundaries, starting and ending, of the packet trace t_i that corresponds to the specific behavior, b_i. One plausible reason is that there may be other control traffic from the IoT device that is being regularly transmitted regardless of the service specific related traffic of the IoT device.

To address this issue, we use a *sliding window* mechanism to extract the features of the network behavior where the term *window* refers to a fixed length sequence of captured packets. The intuition is that as we slide the window over a sequence of the captured network packets, at some point, the exact set of packets that correspond to the behavior will be *covered*. Note that there will still be other unrelated packets within the correct behavioral window, but that is unavoidable in practice.

We illustrate this in Fig. 1, where we consider a network trace of n packets $Pi, \cdots, P_{n+i}$ as the first *window* of behavior under the assumption that a device behavior can be accurately modeled within n packets. The choice of n is subject to practical considerations and complexity of the device. The next window is given by the set of packets $P_{i+1}, \cdots, P_{i+n+1}$, *i.e.*, the next set of n packets starting from the 2^{nd} packet of the first window. Therefore, we do not miss any sequence of n packets, one of which is the most probable candidate covering the specific device-type behavior. Now, the challenge is to define the exact value of n. In our work, we have used the existing experimental data of the IoT device-types to estimate this number and set it to 10. This may not be true for all devices; however, because of the tight coupling of windows, this is a reasonably good estimate of n. Finally, it is important to note that the size of the window,

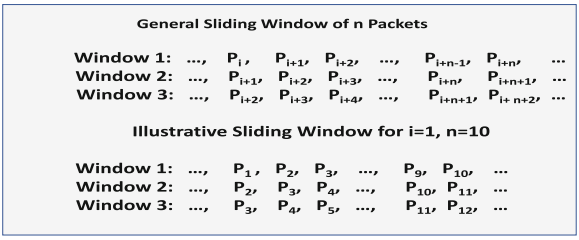

Fig. 1. Sliding window coverage of a specific device-type behavior

n, represents the minimum number of packets that need to be observed from an IoT device-type to verify its registered fingerprint.

3.2 Network Traffic Features of Interest

Now, given a window of packets, the next important step is to extract the network traffic features of interest. Prior research work [4–9,27] has identified various features of interest to fingerprint IoT device-types. We categorize such features into two specific types: protocol and behavioral.

Protocol specific features are used to identify the presence of a particular protocol in a given packet header. These features may only be partially specific to a particular device-type. Sample protocol specific features are: presence of TCP, UDP, HTTPS, HTTP, IPv6, ICMP, and DHCP. These are all binary features, *i.e.*, if the protocol appears in the packet header, the feature is set to 1 otherwise it is set to 0.

Behavior specific features are used to quantify the nature of data or connections within a given window of packets. For instance, the payload lengths of small IoT device-types like smart lights could be considerably different from the payload lengths of a large IoT device-types such as video cameras. Furthermore, some device-types make many connections to neighboring devices while some device-types maintain silence and only communicate when requested. These are behavior specific trends of an IoT device-type and are valuable for generating the fingerprint for the IoT device-type. A few illustrative behavior specific features for a given window of packets are: `payload length`, `payload entropy`, `packet header length`, `the number of unique IP addresses observed`, `the number of distinct sub-net IP addresses`, `number of distinct external destination IP addresses`, `IP address variance`, `length of the longest IP flow`, `number of packets with same source port` and `number of packets with same destination port`. Some of these features are extracted per packet and stored while the other features are extracted from the entire window of packets. As shown in Table 1, these features are numeric features. The intuition

Table 1. Packet Features

Feature	Value
Payload Length	Length in bytes (float)
Payload Entropy	Byte entropy of payload (float)
Header Length	Length of packet header in bytes
Common Source Port	1 if src.port $\leq$ 1023, else 0
Common Destination Port	1 if src.port $\leq$ 1023, else 0
IP Flows	Count of packets with largest number of same source and destination IP
Unique IP Addresses	Number of unique IP addresses in a packet window
IP Addresses in Subnet	Number of IP addresses that belong to the same subnet in a packet window
External IP Addresses	Number of IP addresses that do not belong to the same subnet
IP Variance	Variance in IP addresses
TCP/UDP/HTTP/HTTPS	1 if packet has a protocol field, else 0

for using these features in device-type fingerprinting is that they are likely to have different distributions across different device-types.

3.3 Privacy Preserving Deep Learning Model for Fingerprinting

Now, based on the features identified, we proceed to build a privacy preserving deep learning model for fingerprinting IoT device-types. Unlike the approach in [28], which described the first practical distributed deep learning model, we engineer a neural network design that suits the current problem.

The process considers a group of N-participants in the training process and a global server that maintains an aggregated deep learning model. Each participant maintains a local parameter vector that it updates using its own private training data and from the updates received from the global server. A parameter vector consists of the weights and biases of a respective neural network. The server periodically aggregates the parameter vector updates received from the users and disseminates the information to the participants. The participants then swap out their local parameters for the aggregated parameters that they received from the server and begin the next round of training. After each round of aggregation, all participants and the server will have equivalent parameter vectors. The number of updates and aggregation steps that occur during the process is determined by the number of training samples and two predefined hyper-parameters: $batch\_size$ and $epoch$s. Each participant has the same number of training samples, N, and the swaps occur after each batch so the total number of swaps is: $\frac{N}{batch\_size} * epochs$. Therefore, the parameter aggregation method is a critical factor in the distributed learning process.

Decentralized Optimization. In our distributed learning model, parameter aggregation is done through selective parameter swapping. Participants send the parameters of their local model to the server and the server selects a portion of the parameters in its model to swap out for the ones it received. The selective parameter swapping operation is shown below.

- Server maintains an aggregated parameter vector, w_a
- Server receives parameter vector, w_i, from participant, P_i
- Server computes $index = indices\ of\ sort(|w_i - w_a|)$
- Sets $w_a[index[0 : \Theta]] = w_i[index[0 : \Theta]]$ where Θ is a hyper-parameter that determines how many parameters are swapped out; the server selects the parameters with the greatest absolute difference between the received and aggregated parameter vectors.

We found that the best value for Θ is the length of the parameter vector divided by the number of participants. We call this method "Decentralized Optimization" because, instead of performing parameter optimization on the server, the participants optimize their parameters locally then send their optimized parameters to the server during each round of aggregation. The participants optimize the parameters of their local model using Adam optimization, an extension of Stochastic Gradient Descent, described by [38].

4 Data Synthesis

To address the problem of small data sets, we used an AAE [39] to generate synthetic data for the IoT devices. We define and use the following terminology for describing our AAE.

- *Network fingerprint*: a feature vector extracted from the network traffic of a device.
- *Latent vector*: the compact and hidden representation of the input data, the output of the bottleneck layer of the autoencoder.
- *Prior distribution*: the target distribution of the latent space produced by the encoder given the training data. We used a Gaussian distribution with a mean of 0.5 and standard deviation of 1.0.
- *Reconstruction loss*: the loss function minimized by the autoencoder, we used mean squared error (MSE).

4.1 AAE Architecture

Our AAE consists of the following key components: encoder, decoder, reconstruction discriminator and latent discriminator as shown in Fig. 2. The encoder is a neural network with three hidden layers, which takes the network fingerprint as the input and outputs a 20-by-1 latent vector. The decoder is a neural network with three hidden layers, which takes a 20-by-1 vector as input and outputs either a synthetic or a reconstructed fingerprint (depending on the source of the input vector). The latent discriminator is a neural network classifier with two hidden layers, that takes a 20-by-1 input vector and classifies it into one of two classes: synthetic latent vector (output of encoder), or vector from prior distribution. X represents original network fingerprint samples and is the input to the encoder. The output of the encoder, Z, is a latent vector and has the same dimensions as prior. The latent discriminator takes samples from prior and Z as inputs.

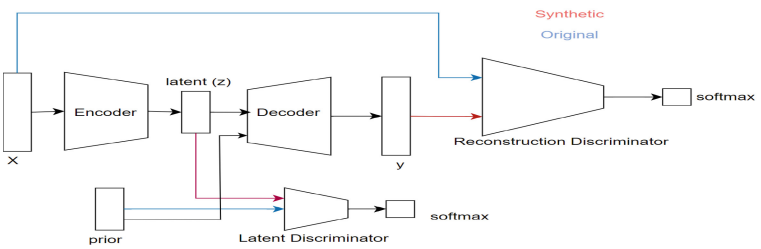

Fig. 2. Adversarial Auto-Encoder

AAE Training. In the first step, of autoencoder update, the encoder and decoder are updated on a mini-batch of samples from the original data. The reconstruction loss from the decoder is back propagated through the decoder and encoder to calculate the gradient. The encoder and decoder weights are updated using the gradient. In the second step, of adversarial update, the mini-batch of samples is fed forward through the encoder to produce the generated latent batch (Z). The latent discriminator gradient is calculated using the latent vector (Z) and a batch from the prior distribution and its weights are updated using the gradient. The batch is fed forward through the encoder and the discriminator to calculate the adversarial error by taking the classification error with the class label inverted. The error is back propagated through the discriminator and the encoder to calculate the gradient. The encoder's weights are updated with the gradient while the discriminator's gradient is discarded.

Adversarial Decoder Training. The decoder training is quite similar as that of the encoder training. In this round, the decoder is fine-tuned to generate believable samples. This round also uses mini-batches from the original data; however, the mini-batches are used as positive samples to train the discriminator. The decoder's weights are updated with the calculated gradient and the discriminator's gradient is discarded.

Synthetic Data Generation. Compared to training the AAE, generating synthetic data samples is straightforward. After the model is trained, the encoder, latent discriminator, and reconstruction discriminator neural networks can all be discarded. The only component that is used to generate data is the decoder. To generate data, the decoder is given a random vector from the prior distribution and it outputs a synthetic vector that is similar to the training data. Given a random vector with shape (num_samples, latent shape), the decoder will produce an output with shape (num_samples, num_features).

5 Performance Evaluation

5.1 Experimental Methodology

We implemented our approach using the numpy library in Python 3.11.4 on a laptop running Windows 11 OS with Intel core® i9-13900H CPU© 2600 MHz processor with 32 GB RAM. We implemented our models using numpy expressions instead of a machine learning framework because this gave us more control. We used a multi-layer perceptron (MLP) as the common neural network architecture for every participant. For training, we used a one-vs-all model of classification where a separate neural network binary classifier was trained for each device-type. Each neural network has a different target vector where the specific device-type was given the positive label while the rest of the device-types were given the negative labels.

All versions of the experiment used a MLP with two hidden layers, sizes 60 and 40. The rectified linear activation function (ReLU) has been used on the hidden layers and softmax on the network outputs. In all the experiments, the batch size is 256 and learning rate is 0.001 with the Binary Cross Entropy (BCE) as the loss function and Adam optimization with 0.9 beta1 and 0.999 beta2 as the parameters. The weights are initialized randomly but any initialization scheme can be used by participants. During the local training, the participants communicate asynchronously with the global parameter server. Once the participant's local model is updated with new weights downloaded from the server, the next training epoch starts. In all the experiments, the value of θ is set to the value (1.0/number of participants).

During classification, we denote a correctly classified positive sample by true positive (TP), an incorrectly classified positive sample as false negatives (FN), a correctly classified negative sample as true negative (TN), and an incorrectly classified negative sample as false positive (FP). We report standard classification metrics as shown in Table 2.

Table 2. Machine Learning Metrics

Metric Name	Definition
Recall (True Positive Rate)	$\frac{TP}{TP+FP}$
Accuracy	$\frac{TP+TN}{TP+FP+FN+TN}$
F1 (F-Score)	$\frac{2TP}{2TP+FP+FN}$
False Positive Rate	$\frac{FP}{FP+TN}$

Communication Model. All of the distributed learning experiments use a single network model. We use a client-server architecture with four clients where the parameter training is done by the clients and the aggregation is done by the server. In our experiments, we simulate dispersed datasets by artificially partitioning the data into five, non-overlapping, subsets of equal size. This partitioning is done by the server using 5-fold cross validation and the server maintains a cumulative sum of confusion matrices of each iteration. We use this confusion matrix to calculate the metrics, which we use to evaluate the model's performance. At the beginning of each iteration, the server partitions the data into four training sets and one test set and sends each client a different training set. Next, the clients send the means and standard deviations for each feature in their training data to the server. Then, the server calculates the combined means and standard deviations for each feature and sends them back to the clients. The clients use the combined means and standard deviations to standardize their training data, which they use to train their local models. At the end of the iteration, the server computes a confusion matrix by testing its aggregated classifier with the data from the test set after standardizing it using the combined means and standard deviations. Training is synchronized so each client sends an

update after every batch then waits for the server to send aggregated parameters back before resuming the training. Also, the server does not send the aggregated parameters to the clients until each client has sent its parameters.

5.2 Data Sets

We collected data sets from a sample set of IoT devices such as Amazon Echo, Philips Smart Light etc., as shown in Table 3. We also used some data collected by one of our team members from WiFi access points (with prior permission), standard access points and taxi-cab hot-spots (data available on request). The access points in cabs are specifically meant for providing communication services like WiFi, GPS and radio. We considered these aspects and treated them as IoT devices. Also, since these devices are slightly more generic than a regular IoT device, they pose some technical challenges in fingerprinting due to the noise in their traffic. In our experiments with data synthesized by the AAE, the training sets included 10% synthetic data while the testing sets only included original data samples.

Table 3. Device Network Data

Device Type	Packets
Amazon Echo	18670
Philips Smart Light	49910
Taxi Cab WiFi APs	49910
Standard WiFi APs	11770
Omna	3110
Somfy	49910

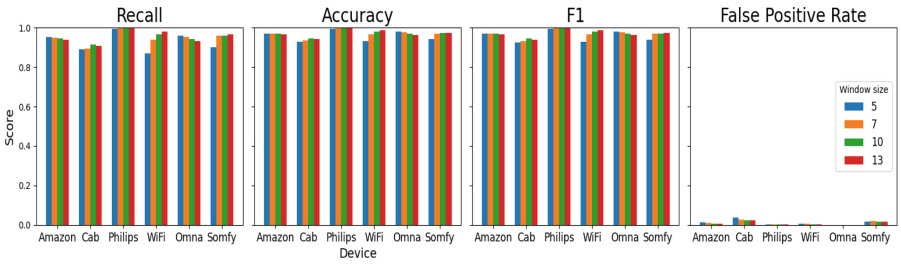

Fig. 3. Packet Window Size Estimation

5.3 Window Size Estimation

One of the difficult choices is the window size of the packets that should be observed to fingerprint a device. Towards this, we measured the performance of our distributed learning model with window sizes 5, 7, 10 and 13. From this experiment, we found that the best value for window size is 10 packets. However, as shown in Fig. 3, the response in the model's performance on individual devices to different window sizes varied. We chose 10 as our estimated window size because it had the best overall performance. The average Recall score with window size 10 was 95.4% which is 2.7% higher than the worst performing window size, 5, and 0.1% higher than the next best window size, 13. We note that identifying the best window size is a difficult task as the device needs to exhibit significant device specific behavior in that window.

5.4 Centralized Learning Performance

To measure the baseline performance, we first performed experiments on the centralized learning mode, *i.e.*, a single party training the machine learning model using all the data.

We compare the performance of the sliding window approach on real and synthetic data. As shown in Fig. 4, our sliding window approach performs well, achieving recall in the range of 83% to 99% across all devices with an average of 92%. The accuracy was in the range of 89% to 99% with an average of 95%. One interesting note is that fingerprinting performance is greater when using synthetic data. This is an encouraging aspect because it shows that the AAE is able to generalize important features in the dataset. If good quality synthetic data is available then it would help other researchers in the community to utilize our method to enhance their experiments as well.

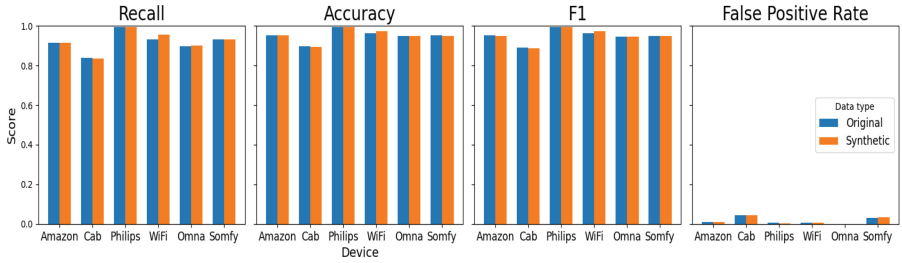

Fig. 4. Baseline Centralized Training Performance On Real and Synthetic Data

5.5 Distributed Learning Performance

The distributed learning method performs well for both real training data and synthetic training data. We tested the distributed learning by training on the

actual data samples and synthetic data samples while the testing was performed real data samples. From Fig. 5, we note that the recall was in the range of 92% to 99% while the accuracy ranged from 95% to 99%. In Fig. 6, we show that our model performs better with four (4) clients than with two (2) clients. Finally, in Fig. 7 we present a comparison of the best results across all the experiments, *i.e.*, the best of centralized against the best of distributed learning model. The distributed model in this case uses four clients and a window size of 10. The distributed model performs better than the centralized model in all metrics. The average recall score for the distributed model is 95.4% which is 3.4% higher

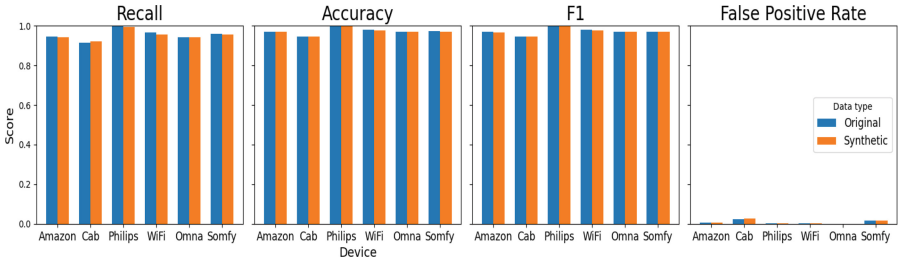

Fig. 5. Distributed Training Model On Real and Synthetic Data

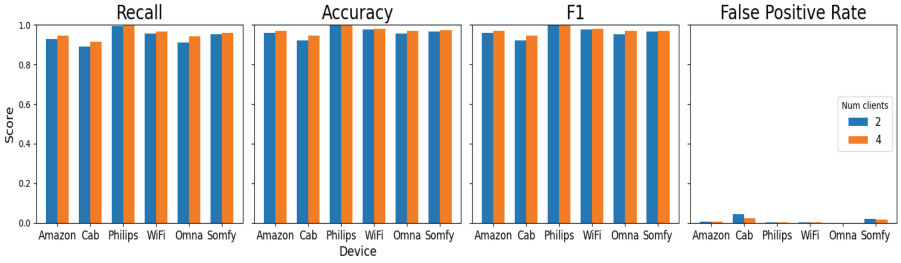

Fig. 6. Distributed Training Model Using 2 and 4 clients

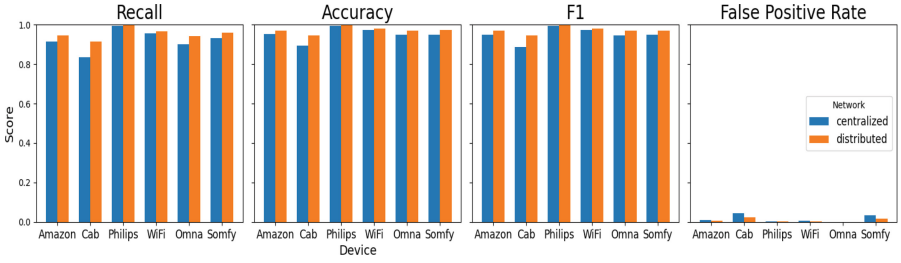

Fig. 7. The Comparison of Best of Distributed and Centralized Fingerprinting Models

than the average recall of the best performing centralized model. One possible reason for this surprising result is that the noise in local data sets seems to be reduced considerably before updating the global neural network model. However, we intend to explore this aspect more rigorously to verify if the same results are achievable for a larger device collection.

6 Conclusion and Future Work

We addressed the problem of fingerprinting IoT devices when sufficient data is not available to the analyst. Our sliding window approach attempts to maximize the chances of covering behavior of device-type and also, increases the packet traffic utilization for machine learning. Our approach generates the fingerprinting model by sharing individual learning model parameters from collaborating parties. We also described an adversarial auto-encoder mechanism to generate synthetic samples that are useful in training the fingerprinting model. Our results show that our approaches perform as good as the centralized learning models with a TPR ranging from 92% to 99% and accuracy ranging from 95% to 99%.

There are several key challenges and future directions for our work. First, the window size of packets is a difficult parameter to estimate as various devices exhibit variable behavior in different packet windows. Second, the IoT devices perform periodic upgrades to their firmware, which might make these devices behave differently at times. Adapting to such changes and evolution in IoT devices is a challenging task as the fingerprinting model might have to learn "on-the-fly". Third, the quality of synthetic data can dramatically change the landscape of IoT fingerprinting. Better synthetic data can resolve the challenges in data collection and analysis and could help the community immensely. Finally, the future scope of our work is to explore the induction of new features into the deep learning model as time evolves and to be able to handle adversarial or noisy data samples from misbehaving participants.

Acknowledgement. This work has been partially supported by funding from the NIST under award number 60NANB18D204, from NSF under award numbers DMS 2123761, CNS 2027750, CNS 1822118 and from the member partner of the NSF IUCRC Center for Cybersecurity Analytics and Automation – Statnett, Cyber Risk Research, AMI, NewPush, and ARL.

References

1. Krebs, B.: Mirai IoT botnet co-authors plead guilty? Krebs on security, November 2017
2. Acar, A., et al.: Peek-a-boo: i see your smart home activities, even encrypted! In: Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks, pp. 207–218 (2020)

3. OConnor, T.J., Mohamed, R., Miettinen, M., Enck, W., Reaves, B., Sadeghi, A.-R.: HomeSnitch: behavior transparency and control for smart home IoT devices. In: Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks, pp. 128–138 (2019)
4. Bai, L., Yao, L., Kanhere, S.S., Wang, X., Yang, Z.: Automatic device classification from network traffic streams of Internet of Things. In: 2018 IEEE 43rd Conference on Local Computer Networks (LCN), pp. 1–9. IEEE (2018)
5. Sivanathan, A., et al.: Classifying IoT devices in smart environments using network traffic characteristics. IEEE Trans. Mob. Comput. **18**(8), 1745–1759 (2018)
6. Dong, S., Li, Z., Tang, D., Chen, J., Sun, M., Zhang, K.: Your smart home can't keep a secret: towards automated fingerprinting of IoT traffic. In: Proceedings of the 15th ACM Asia Conference on Computer and Communications Security, pp. 47–59 (2020)
7. Perdisci, R., Papastergiou, T., Alrawi, O., Antonakakis, M.: IoTFinder: efficient large-scale identification of IoT devices via passive DNS traffic analysis. In: 2020 IEEE European Symposium on Security and Privacy (EuroS&P), pp. 474–489. IEEE (2020)
8. Ahmed, D., Das, A., Zaffar, F.: Analyzing the feasibility and generalizability of fingerprinting Internet of Things devices. Proc. Priv. Enhancing Technol. **2**, 2022 (2022)
9. Sharma, R.A., Soltanaghaei, E., Rowe, A., Sekar, V.: Lumos: identifying and localizing diverse hidden {IoT} devices in an unfamiliar environment. In: 31st USENIX Security Symposium (USENIX Security 2022), pp. 1095–1112 (2022)
10. Franklin, J., McCoy, D.: Passive data link layer 802.11 wireless device driver fingerprinting. In: Proceedings of the 15th USENIX Security Symposium, Vancouver, BC, Canada, 31 July–4 August 2006
11. Pang, J., Greenstein, B., Gummadi, R., Seshan, S., Wetherall, D.: 802.11 user fingerprinting. In: Proceedings of the 13th ACM MOBICOM, pp. 99–110. ACM (2007)
12. François, J., Abdelnur, H.J., State, R., Festor, O.: Automated behavioral fingerprinting. In: Proceedings of the 12th RAID Symposium, pp. 182–201 (2009)
13. Arackaparambil, C., Bratus, S., Shubina, A., Kotz, D.: On the reliability of wireless fingerprinting using clock skews. In: Proceedings of the Third ACM WiSec, pp. 169–174, New York, NY, USA. ACM (2010)
14. Kurtz, A., Gascon, H., Becker, T., Rieck, K., Freiling, F.: Fingerprinting mobile devices using personalized configurations. Proc. Priv. Enhancing Technol. **1**, 4–19 (2016)
15. Martin, A., Doddington, G., Kamm, T., Ordowski, M., Przybocki, M.: The DET curve in assessment of detection task performance. Technical report, National Institute of Standards and Technology Gaithersburg MD (1997)
16. Lippmann, R., Fried, D., Piwowarski, K., Streilein, W.: Passive operating system identification from TCP/IP packet headers. In: Workshop on Data Mining for Computer Security, p. 40 (2003)
17. Kohno, T., Broido, A., Claffy, K.C.: Remote physical device fingerprinting. IEEE Trans. Dependable Secure Comput. **2**(2), 93–108 (2005)
18. Brik, V., Banerjee, S., Gruteser, M., Oh, S.: Wireless device identification with radiometric signatures. In: Proceedings of the 14th ACM MOBICOM, pp. 116–127. ACM (2008)
19. Jana, S., Kasera, S.K.: On fast and accurate detection of unauthorized wireless access points using clock skews. IEEE Trans. Mobile Comput. **9**(3), 449–462 (2010)

20. Radhakrishnan, S.V., Uluagac, A.S., Beyah, R.A.: GTID: a technique for physical device and device type fingerprinting. IEEE Trans. Dependable Secure Comput. **12**(5), 519–532 (2015)
21. Formby, D., Srinivasan, P., Leonard, A., Rogers, J., Beyah, R.A.: Who's in control of your control system? Device fingerprinting for cyber-physical systems. In: 23rd Annual ISOC NDSS (2016)
22. Van Goethem, T., Scheepers, W., Preuveneers, D., Joosen, W.: Accelerometer-based device fingerprinting for multi-factor mobile authentication. In: Caballero, J., Bodden, E., Athanasopoulos, E. (eds.) ESSoS 2016. LNCS, vol. 9639, pp. 106–121. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-30806-7_7
23. François, J., Abdelnur, H.J., State, R., Festor, O.: Machine learning techniques for passive network inventory. IEEE Trans. Netw. Serv. Manage. **7**(4), 244–257 (2010)
24. Gao, K., Corbett, C., Beyah, R.: A passive approach to wireless device fingerprinting. In: Proceedings of IEEE/IFIP DSN, pp. 383–392. IEEE (2010)
25. Miettinen, M., Marchal, S., Hafeez, I., Asokan, N., Sadeghi, A.-R., Tarkoma, S.: IoT SENTINEL: automated device-type identification for security enforcement in IoT. In: Proceedings of 37th IEEE ICDCS, pp. 2177–2184 (2017)
26. Siby, S., Maiti, R.R., Tippenhauer, N.: IoTScanner: detecting and classifying privacy threats in IoT neighborhoods. arXiv preprint arXiv:1701.05007 (2017)
27. Bezawada, B., Bachani, M., Peterson, J., Shirazi, H., Ray, I., Ray, I.: Behavioral fingerprinting of IoT devices. In: Proceedings of the 2018 Workshop on Attacks and Solutions in Hardware Security, ASHES CCS 2018, Toronto, ON, Canada, 19 October 2018, pp. 41–50. ACM (2018)
28. Shokri, R., Shmatikov, V.: Privacy-preserving deep learning. In: 2015 53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton), pp. 909–910 (2015)
29. McMahan, B., Moore, E., Ramage, D., Hampson, S., y Arcas, B.A.: Communication-efficient learning of deep networks from decentralized data. In: Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017, 20–22 April 2017, Fort Lauderdale, FL, USA, vol. 54, pp. 1273–1282. PMLR (2017)
30. Gupta, O., Raskar, R.: Distributed learning of deep neural network over multiple agents. J. Netw. Comput. Appl. **116**, 1–8 (2018)
31. Lévy, D., Jain, A.: Breast mass classification from mammograms using deep convolutional neural networks. CoRR, abs/1612.00542 (2016)
32. Vepakomma, P., Gupta, O., Swedish, T., Raskar, R.: Split learning for health: distributed deep learning without sharing raw patient data. CoRR, abs/1812.00564 (2018)
33. Vepakomma, P., Raskar, R.: Split learning: a resource efficient model and data parallel approach for distributed deep learning. In: Ludwig, H., Baracaldo, N. (eds.) Federated Learning - A Comprehensive Overview of Methods and Applications, pp. 439–451. Springer, Cham (2022)
34. Niyaz, Q., Sun, W., Javaid, A.Y.: A deep learning based DDoS detection system in software-defined networking (SDN). EAI Endorsed Trans. Secur. Saf. **4**(12), 12 (2017)
35. Deval, S.K., Tripathi, M., Bezawada, B., Ray, I.: "X-Phish: Days of Future Past": adaptive & privacy preserving phishing detection. In: 2021 IEEE Conference on Communications and Network Security (CNS), pp. 227–235 (2021)
36. Phong, L.T., Phuong, T.T.: Privacy-preserving deep learning via weight transmission. IEEE Trans. Inf. Forensics Secur. **14**(11), 3003–3015 (2019)

37. Wang, H., Eklund, D., Oprea, A., Raza, S.: FL4IoT: IoT device fingerprinting and identification using federated learning. ACM Trans. Internet Things **4**, 1–24 (2023)
38. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. In: 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, 7–9 May 2015, Conference Track Proceedings (2015)
39. Makhzani, A., Shlens, J., Jaitly, N., Goodfellow, I., Frey, B.: Adversarial autoencoders. arXiv preprint arXiv:1511.05644 (2015)