"Bring your own device!": Adaptive IoT Device-type Fingerprinting using Automatic Behavior Extraction

[Work In Progress Paper]

Maxwel Bar-on
maxbaron@colostate.edu
Colorado State University
Fort Collins, Colorado, USA

Katherine Patterson
kpat1@colostate.edu
Colorado State University
Fort Collins, Colorado, USA

Bruhadeshwar Bezawada
bez.bru@gmail.com
Southern Arkansas University
Magnolia, Arkansas, USA

Indrakshi Ray
indrakshi.ray@colostate.edu
Colorado State University
Fort Collins, Colorado, USA

Indrajit Ray
indrajit@cs.colostate.edu
Colorado State University
Fort Collins, Colorado, USA

Abstract

Internet-of-Things (IoT) is playing a key role in modern society by offering enhanced functionalities and services. As IoT devices may introduce new security risks to the network, network administrators profile the behavior of IoT devices using device fingerprinting. Device fingerprinting typically involves training a machine learning model using the network behavioral data of existing devices. If a new device is added, the network becomes vulnerable to attacks until the time that the machine learning model is trained and updated to integrate the new device. Furthermore, if many devices are regularly added to the network, the cost of adapting the machine learning model can be significant. To address the challenges of security and scalability in fingerprinting, we create a collection of observed behaviors of IoT devices from existing devices and use this collection to construct a fingerprint for a new device. In our approach, we design a bi-component neural network architecture consisting of a transformer-based behavior-extractor (BE) and a fingerprinting interpreter. We perform a one-time training of the BE to extract behaviors from known devices. We use the generated BE for (a) fingerprinting existing devices and (b) adapting the existing fingerprinting model to new device data. In our experiments on 22 diverse IoT devices, we show that our model can identify newly introduced devices as well as known devices with a high identification rate. Our approach improves the time to adapt a model by a factor of 78.3× with no loss of accuracy, achieving recall over 98%.

CCS Concepts

• **Security and privacy** → **Network security**; • **Computing methodologies** → *Machine learning*.

Keywords

IoT, fingerprinting, Transformer, self-supervised learning

SACMAT '25, Stony Brook, NY, USA

ACM ISBN 979-8-4007-1503-7/2025/07
https://doi.org/10.1145/3734436.3734456

ACM Reference Format:
Maxwel Bar-on, Katherine Patterson, Bruhadeshwar Bezawada, Indrakshi Ray, and Indrajit Ray. 2025. *"Bring your own device!"*: Adaptive IoT Device-type Fingerprinting using Automatic Behavior Extraction: [Work In Progress Paper]. In *Proceedings of the 30th ACM Symposium on Access Control Models and Technologies (SACMAT '25), July 8–10, 2025, Stony Brook, NY, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3734436.3734456

1 Introduction

The popularity of the Internet-of-Things (IoT) has created a plethora of attack surfaces in home and organizational networks. This is largely due to the weak built-in security of IoT devices. A new generation of malware [4, 8] specifically targets IoT devices, *e.g.*, like the Mirai malware that compromised over one million IoT devices and launched several major Distributed Denial of Service (DDoS) attacks. In light of such attacks, IoT device fingerprinting, the process of profiling an IoT device from a particular manufacturer, *e.g.*, Philips® Smart LED Light Bulb, is a critical requirement for enforcing the necessary access controls and securing the network.

State-of-the-art fingerprinting approaches [1, 2, 5–7, 9, 10, 12, 13, 16, 19–23, 25] are mostly based on supervised and semi-supervised machine learning. These approaches use machine learning to fingerprint IoT devices. Machine learning models are trained on important features extracted from IoT network traffic. As the IoT industry is rapidly expanding, with new devices emerging regularly, fingerprinting models must constantly adapt to evolving networks. However, adapting an existing fingerprinting model to accommodate new devices is inefficient as the existing model is not reused to create the new fingerprinting model. A major technical challenge in model re-use is that it results in reduced performance when the original network features are insufficient for understanding the unknown behaviors of new IoT devices. Therefore, there is a critical need for efficient methods that provide accurate identification while enabling the re-use of deployed fingerprinting models for rapid adaptation to new devices.

Proposed Approach. To address the challenges of efficient and accuracy-maintaining adaptability in IoT fingerprinting models, we propose a two-stage fingerprinting process: (1) extraction of network behavior patterns from unfiltered samples of traffic and (2) identification of devices based on the extracted patterns. Intuitively

speaking, the "*behavior*" of a device is a comprehension of the traffic flows generated by the device.

To improve efficiency, our approach enables the re-use of the parts of the model responsible for extracting behavior patterns, which account for the majority of the training cost. We achieve this by splitting our fingerprinting architecture into two components. The first component is a transformer [24]-based behavior extractor (BE), which can be re-used when the model is adapted with no additional training. The BE is responsible for the first stage of fingerprinting, extracting relevant behavioral patterns from IoT traffic. The second component is a fingerprint interpreter, which is a neural network multi-classifier responsible for identifying devices. A model can be adapted by training a new fingerprint interpreter, previously learned knowledge is transferred [27] to the adapted model through the re-used BE. This is efficient because the interpreter has a significantly lower storage and computational cost than the BE.

To maintain high accuracy when a model is adapted, our approach leverages a generalized training technique for the BE and a conversational-style feature design. This allows the BE to extract behavioral patterns from devices, even if they were not part of the initial set of devices used to train the BE. Our BE training technique formulates behavior extraction as a separate task from device identification by utilizing self-supervised learning. Instead of learning the identities of the devices used for training the BE, the BE learns to extract behaviors by considering relationships between packets. This allows the BE to extract behavior patterns from devices, even if they were not part of the initial set of devices used to train it. Our feature design is restricted to features that are common across most IoT devices to address the challenge of maintaining network feature robustness as new devices are introduced. Our features include standard features, which encode some fundamental characteristics of packets, and relative features, which capture the complex spatio-temporal conversational view of IoT devices.

Our key contributions are as follows. (i) We describe a transformer based bi-component architecture for IoT device fingerprinting that can efficiently adapt to the addition of new devices. (ii) We describe a conversational-style method for extracting features using the relative communication endpoints, *i.e.*, IP addresses and TCP/UDP port numbers, of packets in a traffic sample to train our behavior-extractor and help understand the "conversational" relationships between packets. (iii) We implement a variety of procedures for training the BE and evaluate their effect on the bi-component architecture's ability to adapt to new devices.

2 Related Work

IoT SENTINEL [17] uses binary classifiers to address the issue of scalability; however, they do not evaluate how the performance of their architecture is affected when new devices are introduced. DeviceMien [18] is an LSTM-based IoT fingerprinting architecture that employs an unsupervised approach for automatic device label assignment for cases where the devices of training data are unknown. IoT-Portrait is an IoT fingerprinting architecture [26] consisting of a transformer encoder and a softmax classifier. They propose a knowledge-distillation approach to efficiently adapt a model to identify unseen devices While this enables model re-use, compared to our approach, it requires more computational resources as the attention layers must be updated to adapt a model. Additionally, the accuracy of the approach used in [26] degrades as the number of new devices increases; whereas, our approach can adapt to increasingly large numbers of new devices while maintaining high performance. ScaNeF-IoT, proposed by Alyaha *et al.* [3], achieves similar performance to IoT-Portrait using an Online Stream Learning classifier. This approach can adapt a trained model for an unseen device by adjusting the existing nodes of the model's decision trees using traffic samples from the new device. The drawback of this approach is that it is susceptible to unpredictable fluctuations in accuracy as new devices are added to the model. AutoIoT [11] introduces a mechanism for adapting fingerprinting models on-the-fly by automatically assigning labels to traffic from unknown devices using k-means clustering, then updating the classifier to identify the new labels. However, the discovered labels may not represent actual devices and provide little value for security policy enforcement. The 30-minute window sampling is another weakness of this approach. Our approach only requires 101 packets to fingerprint a device, which can be captured in a few seconds.

3 Fingerprinting Features

Each input sample for our fingerprinting architecture is a sequence of 32 packets where each packet is represented as a 27-dimensional feature vector: 12 standard features and 15 relative features. To maximize the coverage of our data, we use a sliding-window approach to generate input samples by creating a sample for every sub-sequence, or window, of 32 consecutive packets in each collected traffic trace. *Standard features* provide information about each packet in a sequence and are primarily used as discriminative features to distinguish between traffic from different devices. Our standard features are extracted independency from the headers and payloads of packets and include the following:

- **Subnet Src**: {1 if source IP is in subnet else -1}
- **Subnet Dst**: {1 if destination IP is in subnet else -1}
- **Broadcast Dst**: {1 if destination IP is broadcast else -1}
- **TLS**: {1 if packet includes TLS protocol else -1}
- **TCP**: {1 if packet includes TCP header else -1}
- **UDP**: {1 if packet includes UDP header else -1}
- **HTTP**: {1 if packet uses HTTP port else -1}
- **Common Src**: {1 if packet uses common source port else -1}
- **Common Dst**: {1 if packet uses common dst port else -1}
- **Header Length**: {length of transport-layer header (bytes)}
- **Payload Length**: {length of payload (bytes)}
- **Payload Entropy**: {information entropy of payload}

Relative features act as flow-labels for each packet, which helps evaluate relationships between packets to improve behavior extraction.

3.1 Relative Features

Relative features identify the endpoints of packets using 15 dimensional vectors, giving the model insight into the relationships between packets in a sample. This enhances behavioral comprehension compared to the common approach of manual flow filtering, where each model input contains packets from a single flow. Since device behaviors may involve multi-flows and interactions between

separate flows, flow-based filtering overlooks important relationships between packets that are integral for understanding the behaviors. Further, to avoid overfitting, relative features represent endpoint-identifiers using a local scale defined over finite samples of traffic. Mapping to a local scale introduces overlap between the sets of identifiers associated with each device in the training data, encouraging the model to use endpoint-identifiers for behavior extraction rather than discrimination.

The endpoint-identifiers for a packet are defined as a tuple of four attributes: ⟨*source IP address*, *destination IP address*, *source port*, *destination port*⟩. For a packet $\mathcal{P}^{(i)} \in \mathcal{T}$, where $\mathcal{T}$ is the traffic trace, each attribute $\mathcal{P}^{(i)}_j$ is mapped from a global scale to a local scale, defined over the range of unique values for attribute j within the subsequence $\mathcal{T}^{(i)} \subset \mathcal{T}$. $\mathcal{T}^{(i)}$ contains the 101 packets with the closest timestamps to $\mathcal{P}^{(i)}$ defined as:

$$\mathcal{T}^{(i)} = \begin{cases} \{\mathcal{P}^{(1)}, \ldots, \mathcal{P}^{(101)}\} & \text{if } i \leq 50 \\ \{\mathcal{P}^{(i-50)}, \ldots, \mathcal{P}^{(i)}, \ldots, \mathcal{P}^{(i+50)}\} & \text{if } 50 < i \leq (|\mathcal{T}| - 50) \\ \{\mathcal{P}^{(|\mathcal{T}|-101)}, \ldots, \mathcal{P}^{(|\mathcal{T}|)}\} & \text{if } i > (|\mathcal{T}| - 50) \end{cases}$$

We derive local endpoint-identifiers using a mapping function $R(\mathcal{P}^{(i)}, \mathcal{T}^{(i)}) \rightarrow [1, 101]^4$, which returns a vector containing the index of each endpoint attribute of $\mathcal{P}^{(i)}$ defined over $\mathcal{T}^{(i)}$. The mapping is defined as follows:

$$R(\mathcal{P}^{(i)}, \mathcal{T}^{(i)})_j = \left| \left\{ k \mid k \leq \mathcal{P}^{(i)}_j \wedge k \in \pi_j(\mathcal{T}^{(i)}) \right\} \right|$$

Where $\pi_j(\mathcal{T}^{(i)})$ is a relational-projection that returns the set of unique values for attribute j in $\mathcal{T}^{(i)}$.

The relative features for $\mathcal{P}^{(i)}$ are represented as a vector $Y^{(i)} \in (-1, 1)^{15}$ which is a compression of $R(\mathcal{P}^{(i)}, \mathcal{T}^{(i)})$. This compression is done using an autoencoder $A(x) = d(\mathbb{E}(x))$ with embedding layer $\mathbb{E} : [1, 101]^4 \rightarrow (-1, 1)^{15}$, and decoder $d : (-1, 1)^{15} \rightarrow [1, 101]^4$. After training $A(x)$ for 2,000 epochs, the embedding layer is used to compress the localized endpoint-identifiers into relative feature packets. Relative features $Y^{(i)} \in (-1, 1)^{15}$ for $\mathcal{P}^{(i)}$, with local endpoint vector $Z = R(\mathcal{P}^{(i)}, \mathcal{T}^{(i)})$, are obtained as follows:

$$Y^{(i)}_k = Tanh\left(\sum_j^4 \left(W^{(\mathbb{E})}_{j,Z_j,k} + \beta^{(\mathbb{E})}_{j,k} \right) \right)$$

where $W^{(\mathbb{E})} \in \mathbb{R}^{4\times101\times15}$ and $\beta^{(\mathbb{E})} \in \mathbb{R}^{4\times15}$ are the parameters of $\mathbb{E}$. The parameters of the embedding layer are saved for extracting relative features from new packets during fingerprinting.

4 Fingerprinting Architecture

Our fingerprinting architecture, as shown in Figure 1, consists of two components: a behavior extractor (BE) and an interpreter. The BE is responsible for extracting network behavior patterns from unstructured samples of traffic, while the interpreter is responsible for identifying devices using the extracted patterns. To create a model for fingerprinting, we first train the BE, then we train the interpreter separately using the outputs of the trained BE.

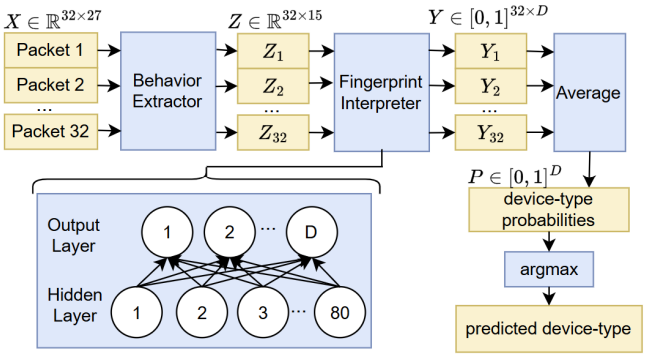

Figure 1: Fingerprinting architecture.

4.1 Automatic Behavior-Extractor

The BE is a vanilla transformer encoder, as described by Vaswani *et al.* [24], having 3 transformer blocks and an embedding dimensionality of 40. For our attention mechanism, we use scaled-dot-product attention with 6 heads. The output of the transformer is projected down to 15-dimensional space using a final linear layer. This encourages generalizability and reduces the size of the interpreter. The transformer extracts behavioral patterns by identifying and encoding relationships between packets in a sample. Given an input sample $X \in \mathbb{R}^{32\times27}$, the BE produces an intermediate representation $Z \in \mathbb{R}^{32\times15}$, which summarizes the temporal and spatial dependencies in the input.

BE training. We train the BE using a training *procedure* consisting of one or more self-supervised learning tasks. In this work, we evaluate several different training procedures. Our self-supervised tasks include *anomaly-locating*, *denoising*, and *masked-autoencoding*.

Training on self-supervised tasks allows the BE to learn general behavioral patterns, allowing it to effectively extract behaviors from new devices. For each task, we train a self-supervised model $g(x) = f(BE(x))$ consisting of the BE followed by a task-specific interpreter f, which is a fully-connected neural network with a single 80-dimensional hidden layer and *ReLU* activation. We update the BE by back-propagating losses through the interpreter to calculate the loss gradient with respect to the BE's learnable parameters. The gradient *W.R.T.* the BE's parameters θ_{be} are calculated as:

$$\frac{d\mathcal{L}}{d\theta_{be}} = \frac{d\mathcal{L}}{df}\frac{df}{d\theta_{be}}$$

where $\mathcal{L}$ is the loss function for the task. See Appendix 8.1 for details on the self-supervised tasks.

To apply a procedure, we start by initializing the learnable parameters of the BE and the interpreters for each of task in the procedure using Xavier initialization [14]. We then combine the BE and the interpreters to create a single model where each task has a designated path that spans the BE and its interpreter so that the BE is shared among all of the paths. We also assign a learning rate to each path with an initial value of 0.005. During the procedure, we iteratively optimize the parameters along each path by minimizing the loss function for its associated task on the training data using *Adam* optimization [15]. After each epoch, we evaluate the loss function of each path using a validation dataset. If the validation loss for a path does not improve for 15 epochs, we reduce its learning rate by

a factor of 3×. If the learning rate for a path degrades below 0.0001, we remove it from the combined model and discard the interpreter. The procedure terminates once all paths are removed.

4.2 Fingerprint Interpreter

The interpreter in the fingerprinting model is a fully-connected neural network multi-classifier with a single 80-dimensional hidden layer, *ReLU* activation, and a sequence-wise average pooling/softmax layer. Given an intermediate representation $Z \in \mathbb{R}^{32\times15}$ from the BE, the interpreter produces a probability distribution $Y \in [0,1]^{|D|}$ over the set of devices, D. Let $Y' \in \mathbb{R}^{32\times|D|}$ be the output of the final linear layer of the interpreter. The pooling layer converts this to a probability distribution as follows:

$$Y_i = \frac{1}{32}\sum_{j}^{32}\frac{exp\left(Y'_{j,i}\right)}{\sum_{k}^{|D|} exp\left(Y'_{j,k}\right)}$$

Here, Y_i represents the probability that behavior-extraction Z was generated by device D_i: $Y_i = P(D_i|Z)$. See Appendix 8.2 for details on the fingerprint interpreter training.

IoT Fingerprinting. After training the BE and interpreter, we create a pipeline model $g(X) = f(BE(X))$ for fingerprinting devices. To fingerprint a sample of traffic $X \in \mathbb{R}^{32\times27}$, we apply the pipelined model to predict a probability distribution over the set of devices $Y \in [0,1]^{|D|}$: $Y = g(X)$. We return the device with the highest corresponding probability in Y.

5 Adapting to New Devices

To adapt the model, we initialize a new fingerprint interpreter and train it to identify an updated set of devices. Let $D' = D \cup D^{new}$ be the updated set of devices, where D is the set of known devices and D^{new} is the set of newly introduced devices. The new fingerprint interpreter $f'(Z) \rightarrow [0,1]^{|D'|}$ has an expanded output dimensionality of $|D'|$. To train f', we collect traffic from the devices in D^{new}, then apply the BE to the collected traffic to generate new intermediate representations. We train f' on the combined intermediate representations from D and D^{new}. Finally, we replace the current fingerprinting model with the updated pipeline: $g(x) = f'(BE(X))$. Notably, for the devices in our data-set, adapting does not require adjusting the BE, as its initial self-supervised training enables adaptation to new devices. With this approach, due to the smaller size and lower computational complexity of the fingerprint interpreter, adapting a fingerprinting model is significantly more efficient than training an equivalent static model.

6 Results

We ran our experiments using Python 3.9.18 and cuda version 11.8 on a desktop with Intel® Xeon® CPU E5-1650 v4 3.60GHz and an NVIDIA TITAN V© GPU with 12288MiB memory. We implement our deep-learning architecture using jax version 0.4.23 for cuda. Jax provides functionality for automated differentiation and just-in-time compilation using XLA. We evaluated our models using 5-fold cross-validation with a train:validation:test ratio of 7:1:2.

In our experiments, we use a 'static' fingerprinting model as a baseline for comparing the performance and efficiency of our adaptable model. The static model has the same size and components as the adaptable model; however, the BE and the fingerprint interpreter are trained jointly in one shot for device identification. We train it using the same optimization settings as the fingerprint interpreter discussed in Section 4.2. We evaluate the following procedures for training the BE component of the adaptable model:

- L+M: anomaly locating and masked autoencoding
- Denoiser: only denoising
- L+D: anomaly locating and denoising
- M+D: masked autoencoding and denoising
- Mask: only masked autoencoding

Adaptation Efficiency. Here, we evaluate the efficiency of our adaptable fingerprinting approach in terms of the time required to adapt a model when new devices are introduced to a network and explore how the procedure used to train a BE impacts the adapt time. For each procedure, we measure the amount of time required to train a new interpreter until it achieves performance on-par with a static model. To reflect networks without access to internal GPUs, we measure the adapting time on CPUs as well as GPUs. In Table 1, we compare the adaptation efficiency of the adaptable model with each procedure against the static model. In both environments, the adaptable model achieves a faster adapting time than the static model for all BE-training procedures. Of the five procedures, 'Denoiser' results in the fastest adapting time followed closely by L+D. On the GPU, this model can be adapted 78.3× faster than the static model, while on the CPU, it is 21.7× faster. The improvement is greater on the GPU because the adaptable model reduces the number of synchronization steps required to avoid overflowing the GPU memory during gradient calculation.

Table 1: Adapting Time.

	Adaptable Models					Static Model
	L+M	Denoiser	L+D	M+D	Mask	
GPU	384	60	67	303	391	4,698
CPU	22,921	3,619	4,021	18,095	23,323	78,513

Performance. Here, we evaluate the capability of our architecture to adapt to new devices. In these experiments, we withhold some devices from the dataset before training the BE, then re-introduce them and adapt the model to identify them. The devices that are withheld represent new devices that the model must adapt to identify and the devices that are used to train the BE represent the set of initial devices for a model. We run many trials with different sets of initial and new devices to thoroughly explore the performance of our adaptable model. The goal of this exploration is to compare how different BE-training procedures affect a model's ability to adapt to changing network requirements.

We evaluate our architecture with varying numbers of initial devices to demonstrate its ability to adapt in various scenarios. Cases with fewer initial devices demonstrate more challenging scenarios since there are fewer examples that the BE can pull from to learn how to extract behaviors. We evaluate our model with 4, 7, 10, 13, 16, and 19 initial devices. For each amount, we repeatedly evaluate our model with different combinations of initial devices until we have measured the performance for each device as an initial device, and a new device, at least three times.

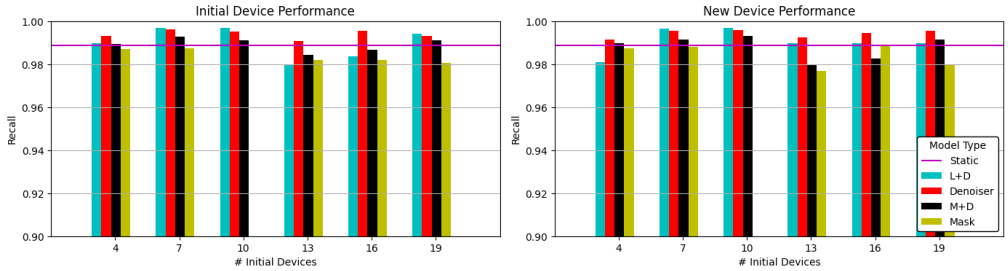

Figure 2: Average recall of different fingerprinting models with different initial devices. The static model is the horizontal line.

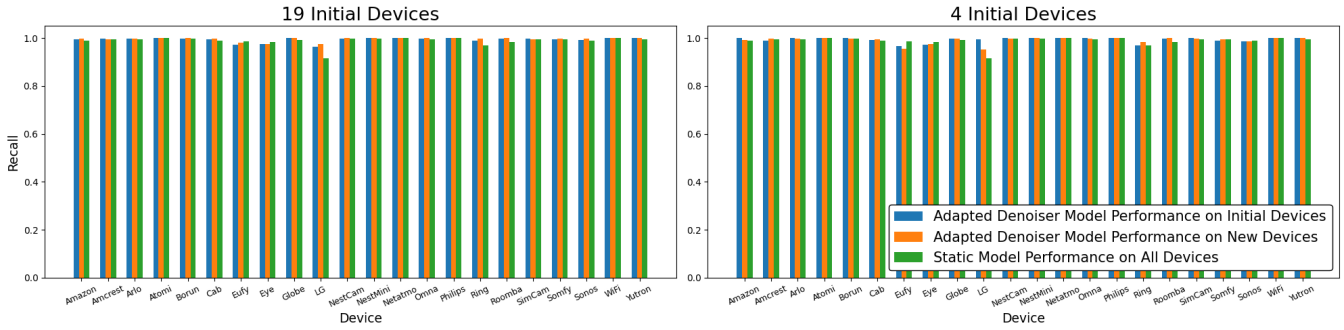

Figure 3: Average recall results of the Denoiser adapted model vs. the static model for all devices.

In Figure 2, we compare how changing the number of initial devices affects the performance of adaptable models with different BE-training procedures. In all scenarios, the four adaptable models achieve a high identification rate, above 97%, for the initial devices as well as new devices. With all procedures, the model can reliably identify new devices with very few initial devices for training the BE. This shows that the BE is capable of learning general patterns for extracting behaviors, even if it is only exposed to the specific behaviors of a small number of initial devices during BE training. While they still achieve a high identification rate for new and initial devices, the "Mask" and "M+D" models perform worse in some scenarios compared to the other models. This suggests that the masked-autoencoding task (masking) is less reliable than the denoising task, as procedures that include masking result in less stable performance across scenarios. Of the four, the "Denoiser" model is the only adaptable model that consistently outperforms the static model, both on initial and new devices. Although the "L+D" model performs better with 7 or 10 initial devices, its performance is less consistent across all scenarios. In Figure 3, we show the performance of the fingerprinting model, trained using the denoiser procedure, for each individual device with the most and the least number of initial devices. We show the performance for each device when it is included in the set of initial devices and when it is included in the set of new devices. In both scenarios, this fingerprinting model identifies every device at a rate that is on-par with or better than the static model.

7 Conclusion and Future Work

In this work, we demonstrate that our IoT fingerprinting approach successfully addresses the problem of efficiently adapting an existing model to identify new IoT devices. By combining relative features with standard features for analyzing network-traffic, our architecture achieves an average base device identification rate of 98.86% for 22 unique IoT devices. With our approach, adapting a fingerprinting model is over 78× faster when compared to the traditional approach. We found that the adapted model achieved an average identification rate above 98% with as many as 81% new devices introduced after the initial training. We explore a variety of procedures for training the behavior extractor component of our architecture and found that the denoiser procedure was the most reliable and effective. Future work can explore different BE and interpreter architectures to further improve the adaptability and performance. Additional work is necessary to understand automatic behavior extraction, since this work does not investigate the values produced by the BE and only focuses on the quality of the interpreter's predictions given the BE's output.

Acknowledgement

This work was partially supported by NSF under Grant No. CNS 1822118, CNS 2226232, DMS 2123761, by the industry member partners of the NSF IUCRC Center for Cyber Security Analytics and Automation (AMI, NewPush, Cyber Risk Research, NIST and ARL), by the State of Colorado (grant #SB 18-086), by NIST Grant No. 60NANB23D152 and by the authors' institutions.

References

[1] Abbas Acar, Hossein Fereidooni, Tigist Abera, Amit Kumar Sikder, Markus Miettinen, Hidayet Aksu, Mauro Conti, Ahmad-Reza Sadeghi, and Selcuk Uluagac. 2020. 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*. 207–218.

[2] Dilawer Ahmed, Anupam Das, and Fareed Zaffar. 2022. Analyzing the feasibility and generalizability of fingerprinting Internet of Things devices. *Proceedings on Privacy Enhancing Technologies* 2022, 2 (2022).

[3] Tadani Nasser Alyahya, Leonardo Aniello, and Vladimiro Sassone. 2024. ScaNeF-IoT: Scalable Network Fingerprinting for IoT Device. In *Proceedings of the 19th International Conference on Availability, Reliability and Security* (Vienna, Austria) *(ARES '24)*. Association for Computing Machinery, New York, NY, USA, Article 195, 9 pages.

[4] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J Alex Halderman, Luca Invernizzi, Michalis Kallitsis, et al. 2017. Understanding the mirai botnet. In *26th USENIX security symposium (USENIX Security 17)*. 1093–1110.

[5] Lei Bai, Lina Yao, Salil S Kanhere, Xianzhi Wang, and Zheng Yang. 2018. Automatic device classification from network traffic streams of internet of things. In *2018 IEEE 43rd conference on local computer networks (LCN)*. IEEE, 1–9.

[6] Maxwel Bar-on, Bruhadeshwar Bezawada, Indrakshi Ray, and Indrajit Ray. 2024. A Small World–Privacy Preserving IoT Device-Type Fingerprinting with Small Datasets. In *Foundations and Practice of Security*, Mohamed Mosbah, Florence Sedes, Nadia Tawbi, Toufik Ahmed, Nora Boulahia-Cuppens, and Joaquin Garcia-Alfaro (Eds.). Springer Nature Switzerland, Cham, 104–122.

[7] Bruhadeshwar Bezawada, Maalvika Bachani, Jordan Peterson, Hossein Shirazi, Indrakshi Ray, and Indrajit Ray. 2018. Behavioral Fingerprinting of IoT Devices. In *Proceedings of the 2018 Workshop on Attacks and Solutions in Hardware Security, ASHES@CCS 2018, Toronto, ON, Canada, October 19, 2018*. ACM, 41–50.

[8] Andrei Costin and Jonas Zaddach. 2018. Iot malware: Comprehensive survey, analysis framework and case studies. *BlackHat USA* 1, 1 (2018), 1–9.

[9] Shuaike Dong, Zhou Li, Di Tang, Jiongyi Chen, Menghan Sun, and Kehuan Zhang. 2020. 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* (Taipei, Taiwan) *(ASIA CCS '20)*. Association for Computing Machinery, New York, NY, USA, 47–59.

[10] Chenxin Duan, Shize Zhang, Jiahai Yang, Zhiliang Wang, Yang Yang, and Jia Li. 2021. PINBALL: Universal and Robust Signature Extraction for Smart Home Devices. In *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. 1–9.

[11] Linna Fan, Lin He, Yichao Wu, Shize Zhang, Zhiliang Wang, Jia Li, Jiahai Yang, Chaocan Xiang, and Xiaoqian Ma. 2023. AutoIoT: Automatically Updated IoT Device Identification With Semi-Supervised Learning. *IEEE Transactions on Mobile Computing* 22, 10 (2023), 5769–5786.

[12] Justin Feng, Tianyi Zhao, Shamik Sarkar, Dominic Konrad, Timothy Jacques, Danijela Cabric, and Nader Sehatbakhsh. 2023. Fingerprinting IoT Devices Using Latent Physical Side-Channels. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 7, 2, Article 54 (June 2023), 26 pages.

[13] Jérôme François, Humberto J. Abdelnur, Radu State, and Olivier Festor. 2009. Automated Behavioral Fingerprinting. In *Proc. of the 12th RAID Symposium*. 182–201.

[14] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 249–256.

[15] Diederik P. Kingma and Jimmy Ba. 2017. Adam: A Method for Stochastic Optimization. arXiv:1412.6980 [cs.LG]

[16] Manuel Lopez-Martin, Belén Carro, Antonio Sanchez-Esguevillas, and Jaime Lloret. 2017. Network Traffic Classifier With Convolutional and Recurrent Neural Networks for Internet of Things. *IEEE Access* PP (09 2017), 1–1.

[17] Markus Miettinen, Samuel Marchal, Ibbad Hafeez, N. Asokan, Ahmad-Reza Sadeghi, and Sasu Tarkoma. 2017. IoT SENTINEL: Automated Device-Type Identification for Security Enforcement in IoT. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. 2177–2184.

[18] Jorge Ortiz, Catherine Crawford, and Franck Le. 2019. DeviceMien: network device behavior modeling for identifying unknown IoT devices. In *Proceedings of the International Conference on Internet of Things Design and Implementation* (Montreal, Quebec, Canada) *(IoTDI '19)*. Association for Computing Machinery, New York, NY, USA, 106–117.

[19] Roberto Perdisci, Thomas Papastergiou, Omar Alrawi, and Manos Antonakakis. 2020. Iotfinder: Efficient large-scale identification of iot devices via passive dns traffic analysis. In *2020 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 474–489.

[20] Chao Shang, Jin Cao, Tong Zhu, Yinghui Zhang, Ben Niu, and Hui Li. 2024. CADFA: A Clock Skew-Based Active Device Fingerprint Authentication Scheme for Class-1 IoT Devices. *IEEE Systems Journal* 18, 1 (2024), 590–599.

[21] Sandra Siby, Rajib Ranjan Maiti, and Nils Tippenhauer. 2017. IoTScanner: Detecting and Classifying Privacy Threats in IoT Neighborhoods. arXiv:1701.05007 [cs.CR]

[22] Arunan Sivanathan, Hassan Habibi Gharakheili, Franco Loi, Adam Radford, Chamith Wijenayake, Arun Vishwanath, and Vijay Sivaraman. 2018. Classifying IoT devices in smart environments using network traffic characteristics. *IEEE Transactions on Mobile Computing* 18, 8 (2018), 1745–1759.

[23] Jay Thom, Nathan Thom, Shamik Sengupta, and Emily Hand. 2022. Smart Recon: Network Traffic Fingerprinting for IoT Device Identification. In *2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)*. 0072–0079.

[24] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

[25] Han Wang, David Eklund, Alina Oprea, and Shahid Raza. 2023. FL4IoT: IoT Device Fingerprinting and Identification Using Federated Learning. *ACM Trans. Internet Things* (jun 2023).

[26] Juan Wang, Jing Zhong, and Jiangqi Li. 2023. IoT-Portrait: Automatically Identifying IoT Devices via Transformer with Incremental Learning. *Future Internet* 15, 3 (2023).

[27] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. 2021. A Comprehensive Survey on Transfer Learning. *Proc. IEEE* 109, 1 (2021), 43–76.

8 Appendix

8.1 BE Training Tasks

In the *masked-autoencoding* task, we apply a binary mask to the packet features and train the model to reconstruct the missing values. For each input sample, we randomly generate a mask $M \in \{0,1\}^{32\times27}$ from a binomial distribution where each value has a 60% probability of having the value 1. The output of the model $Y \in \mathbb{R}^{32\times27}$ is defined as $Y = g(M \odot X)$ where $\odot$ denotes element-wise multiplication. We train the model to minimize the mean-squared-error (MSE) loss between X and Y.

The *denoising* task is the same as the *masked-autoencoding* task; however, instead of applying a binary mask, we add a noise vector $\mathcal{U} \in (-0.15, 0.15)^{32\times27}$ sampled from a uniform distribution to the input. The output of the model $Y \in \mathbb{R}^{32\times27}$ is defined as $Y = g(X \oplus \mathcal{U})$ where $\oplus$ denotes element-wise addition.

In *anomaly-locating*, we randomly insert an anomalous packet into each window and train the model to predict its position relative to each packet. For each input sequence $X \in \mathbb{R}^{32\times27}$, we randomly select one row a, and replace it with a random vector $\mathcal{A} \in [-1,1]^{27}$. We implement this a 3-class classification problem where, for each packet X_i, the model predicts the most likely option from the set of classes $C = \{i < a, i = a, i > a\}$. The interpreter includes a $softmax$ operation following the final linear layer, and outputs a probability matrix $Y \in [0,1]^{N\times32\times3}$. We train the model to minimize the negative-log-likelihood loss between Y and the true labels for each packet from C.

8.2 Interpreter Training Details

Compared to the BE, the fingerprint interpreter is lightweight and can be trained efficiently on GPUs or CPUs. We train the interpreter to identify devices corresponding to the intermediate representations produced by the trained BE. Throughout the entire training process, the training and validation data only requires one forward pass through the BE to produce intermediate representations. We train the interpreter using *Adam* to minimize the negative-log-likelihood loss between the output probabilities Y, and the true device labels from D. We use an initial learning rate of 0.03 and decay rate of 3×.