iPET: Privacy Enhancing Traffic Perturbations for Secure IoT Communications

Akshaye Shenoi*
National University of Singapore
Singapore
akshaye@comp.nus.edu.sg

Prasanna Karthik Vairam*
National University of Singapore
Singapore
prasanna@comp.nus.edu.sg

Kanav Sabharwal*
National University of Singapore
Singapore
kanav.sabharwal@u.nus.edu

Jialin Li
National University of Singapore
Singapore
lijl@comp.nus.edu.sg

Dinil Mon Divakaran
Acronis Research
Singapore
dinil.divakaran@acronis.com

ABSTRACT

IoT devices constantly communicate with servers over the Internet, allowing an attacker to extract sensitive information by passively monitoring the network traffic. Recent research works have shown that a network attacker with a trained machine learning (ML) model can accurately fingerprint IoT devices learned from the (encrypted) traffic flows of the devices. Such fingerprinting attacks are capable of revealing the make and model of the devices, which can further be used to extract detailed user activities.

In this work, we develop and propose iPET, a novel adversarial perturbation-based traffic modification system that defends against fingerprinting attacks. iPET design employs GAN (Generative Adversarial Networks) in a tuneable way, allowing users to specify the maximum bandwidth overhead they are willing to tolerate for the defense. A fundamental idea of iPET is to deliberately introduce stochasticity between model instances. This approach limits a counter attack, as it inhibits an attacker from recreating an identical perturbation model and using it for fingerprinting. We evaluate the effectiveness of our defense against state-of-the-art fingerprinting models and with three different attacker capabilities. Our evaluations on synthetic and real-world datasets demonstrate that iPET decreases the accuracy of even the potent attackers. We also show that the traffic perturbations generated by iPET generalize well to different fingerprinting schemes that an attacker may deploy.

KEYWORDS

IoT, privacy, fingerprinting, deep learning, adversarial machine learning

1 INTRODUCTION

Internet of Things (IoT) devices are fast penetrating into different markets such as consumer, enterprise, critical infrastructure, education, and healthcare. The number of IoT devices in the world is projected to go up to 41 billion by 2027, a steep rise from 8 billion in 2019 [33]. IoT devices have become part of our daily lives; we use them to track and monitor heart beat rate, sleep quality, sports activities, etc., as well as for smart, simplified and automated operations such as controlling home environment and ordering groceries. This unprecedented growth is, however, accompanied with increasing threats and attacks [28]. In particular, user privacy is vulnerable to eavesdropping attack — a network attacker having access to the communication traffic of IoT devices can identify the device types and even device activities.

Advances in machine learning (ML) have largely benefited our society; unsurprisingly, they have also become a useful tool for attackers. Many recent attacks leverage machine learning models to target the privacy vulnerabilities of IoT devices [3, 10, 13, 27, 44, 47, 53]. These ML-based *device fingerprinting attacks* are capable of identifying devices based on the communication patterns captured by a trained model, even when the payloads are encrypted. This leads us to an important question – how can we protect the privacy of IoT users from network attackers that use state-of-the-art ML-based fingerprinting attacks? While a possible solution would be to modify the traffic by adding perturbations that are random or by maintaining a constant traffic rate, such approaches are either ineffective or incur high bandwidth overhead (Section 5.1.2).

Recently, researchers have shown that small changes in the inputs — *adversarial perturbations* — can effectively deceive trained classification models [14, 30, 36, 51]. The properties of adversarial perturbations i) of being small in magnitude, ii) fooling a classifier to force misclassifications, iii) and generalizing between models, make them suitable for a traffic obfuscation mechanism to counter device fingerprinting attacks.

Applying these defense perturbations to network traffic, however, has its challenges: i) ensure the perturbations are within the network constraints and do not disrupt device communications, ii) minimize the cost of defense in terms of network bandwidth and latency, and most importantly iii) make the defense robust to adversarial training (wherein adversary retrain their classifier with the knowledge of the defense deployed by a user) [14, 22, 54].

In this work, we overcome the above challenges by designing and developing iPET, a privacy-enhancing technique for IoT devices, that leverages adversarial ML to perturb network traffic and thereby defeat fingerprinting attacks. iPET employs a generative deep learning model to produce device specific defense perturbations. The

*Authors contributed equally.

This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license visit https://creativecommons.org/licenses/by/4.0/ or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.
Proceedings on Privacy Enhancing Technologies 2023(2), 206–220
© 2023 Copyright held by the owner/author(s).
https://doi.org/10.56553/popets-2023-0048

model is independent of live network traffic of communicating devices, allowing the perturbations to be pre-generated, thereby minimizing the network latency. We design iPET as a tuneable defense, allowing users to choose the maximum bandwidth overhead acceptable to them. An important aspect of iPET is that it is resilient to adversaries retraining their models based on the understanding of the defense. The key idea is a training process that ensures the iPET instances created, and thereby the perturbation patterns generated at the defense (and also by an adversary), are made to diverge significantly.

To the best of our knowledge, iPET is the first solution to use adversarial ML to protect privacy of IoT users.

We summarise the contributions of our work below.

- We develop and present iPET, a novel technique that exploits the stochasticity between instances of generative models to add adversarial perturbations in a way that undermines ML-based device fingerprinting models.
- iPET incurs low overhead in terms of network bandwidth and latency, to achieve the privacy goal of averting device fingerprinting attacks. At the same time, the tuneable nature of iPET allows users to limit the maximum network overhead they are willing to tolerate.
- We conduct comprehensive evaluations using synthetic data as well real network traces of IoT devices. The experiments demonstrate that iPET is effective against an attacker using different fingerprinting architectures. We also compare iPET against existing traffic obfuscation techniques in the literature and show that iPET is more effective and resilient to powerful adversaries in countering device fingerprinting attacks at a much lower network bandwidth overhead. Going further, our evaluations consider an attacker with different capabilities, with the most powerful attacker having complete knowledge of the defense model. Finally, we show that our approach is transferable across the defense's configuration parameters and effective in the open world.
- We implement iPET in Python using Keras libraries from TensorFlow [52], and publish the source code online[1].

In the following section, we present the background necessary for our work and formally define the threat model. In Section 3, we present the goals and challenges in developing a defense solution, and provide an overview of iPET. We develop iPET in Section 4 and analyze its resilience to various attack scenarios. In Section 5, we pose important research questions on countering device fingerprinting, and evaluate iPET in this context. Section 6 opens up further discussions on iPET. Related works are discussed in Section 7.

2 BACKGROUND AND THREAT MODEL

2.1 System: Users and Attacker

We consider a system where IoT devices communicate with different servers on the Internet for routine operations. IoT devices connect to the Internet often with the help of a gateway (e.g., home IoT gateway) as shown in Figure 1. The potential victims here are the users of IoT devices, who are concerned about privacy leaks on the Internet links, despite the communications being encrypted.

[1]https://github.com/akshayeshenoi/ipet

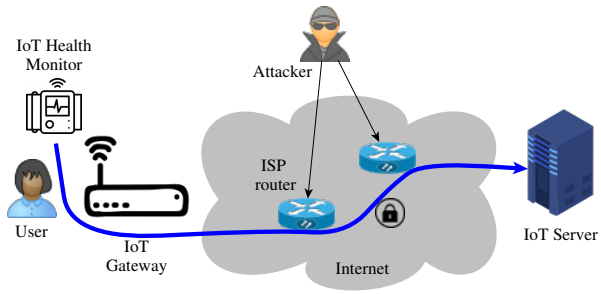


Figure 1: Threat model: an IoT device communicates with a server in the Internet; the attacker could be any router on the communication path with access to (encrypted) traffic exchanged between the IoT devices and their servers.

We assume that an attacker may control any network device (e.g., router) on the communication path between an IoT device and its server, except the user gateway and the server, which are assumed to be secured and hardened. The attacker may do so by compromising a network device (e.g., a router) belonging to an Internet Service Provider (ISP), or the attacker could just be a malicious ISP. The attacker is capable of passively eavesdropping on both the incoming and outgoing communications at the network device. However, the attacker does not have access to home networks of users and their internal traffic; we consider such attacks out of scope of the current work. We also assume that DDoS attacks on gateways or IoT devices (e.g., see [49]) are out of scope of our work, since we consider only a passive attacker and attacks on user privacy.

2.2 Attacker: Goals and Capabilities

The attacker's objective is to infer sensitive information about the users of the IoT devices from the traffic captured at the network entity under his control. The attacker uses a machine learning (ML) classifier to carry out *device fingerprinting attack* — a network attack that infers the type (make and model) of the IoT device [3, 10, 13, 27, 44, 47, 53]. Device fingerprinting could be the first step of more serious attacks, such as those that infer fine-grained sensitive information such as IoT user activity [2]. To build this ML classifier, the attacker may purchase his own IoT devices, generate (potentially encrypted) traffic from them, label them to subsequently extract *features* and train a device-fingerprinting model. This trained model is employed on traffic of users to classify and identify IoT devices. Since packet payloads are often encrypted (e.g., due to the increasing adoption of TLS [15]), most of the features for an ML classifier come from the header fields of the packets. Features may be extracted for each connection (e.g., HTTP connection), or for a session (defined by static or dynamic time-window) of a device. To be specific, we define two broad classes of fingerprinting models for identifying IoT devices:

i) $\mathcal{M}_{\mathbf{agg}}$**: Model Based on Feature Aggregates.** The features of this model are extracted from sessions of fixed-length time-windows of a device. Existing works select time-windows ranging in minutes to achieve high fingerprinting accuracy (e.g., DEFT [53] uses a

window length of 15 minutes). For a given session, the features include, the number of packets, statistics (mean, minimum, maximum, and sum) of the packet lengths, etc. The features are aggregated separately for incoming and outgoing packets of a device (IP address) and further grouped by the different protocols (e.g. DNS, HTTP, NTP, etc.). Appendix A gives the full list of features. Machine learning classifiers such as Random Forest [5] can be used for training.

ii) $\mathcal{M}_{\mathbf{seq}}$**: Sequence-Based Device Classifier.** With this model, the goal is to capture the temporal relationships between packets in a device's traffic session. For readability, we denote the features of a session as a matrix τ of shape $[f, n]$, where each of the n columns corresponds to a discrete time-slot ω_i (say, of length 100ms) in a session. Essentially, τ captures the session as a time-series; e.g., if session-length is 1s, there would be 10 columns in τ. We consider f features (rows of τ); they represent the number of incoming/outgoing packets in ω_i, the sum of lengths of incoming/outgoing TCP payloads in ω_i, and the sum of lengths of incoming/outgoing UDP payloads in ω_i. Although these features appear simple, they are known to be effective in fingerprinting IoT devices [13, 55]; besides they can be extracted with minimal processing. Additionally, we compute the silence period s between two consecutive sessions (time between the first packet of current session and last packet of previous session) as another feature. Deep Neural Network (DNN) architectures such as LSTM [17], CNN [21], GRU [7], and MLP [40] can be used to train such models.

We note that $\mathcal{M}_{\text{agg}}$ relies on packet header information (e.g., application layer protocols) for computing aggregated features. However, such features are unavailable when anonymous communication tools are used. For instance, Tor uses an overlay network that routes encrypted user packets over relay nodes to hide the packet headers. In such scenarios, we find that the classification accuracy of $\mathcal{M}_{\text{agg}}$ reduces significantly due to the limited features available. On the contrary, $\mathcal{M}_{\text{seq}}$ is expected to perform better than $\mathcal{M}_{\text{agg}}$ since it attempts to capture the sequence of features (e.g., number and sizes of packets) in a window rather than aggregating them. In Appendix B, we evaluate the accuracy of $\mathcal{M}_{\text{seq}}$ and $\mathcal{M}_{\text{agg}}$ against Tor, consequently conclude that $\mathcal{M}_{\text{seq}}$ is the stronger fingerprinting model. Therefore, we only consider $\mathcal{M}_{\text{seq}}$ in all our evaluations.

Furthermore, we endow our attacker with the knowledge of any defense that an IoT device user may have in place. For instance, when a user employs Tor for defense, we assume that an attacker would be able infiltrate the overlay network and take control of a Tor relay node. Similarly, if the user employs a traffic perturbation technique for defense, we assume that the attacker may deploy the defense at his own premises, route the traffic of his devices through the defense, obtain the perturbed traffic and subsequently retrain his ML classifier. We elaborate the attack scenarios in Section 4.1.

3 GOALS, CHALLENGES, AND OVERVIEW

3.1 Defense Goal and Challenge

Consider the communications between an IoT BP Monitor and its server through the network path, namely, IoT Device $\rightarrow \mathcal{R}_0 \rightarrow \mathcal{R}_1 \rightarrow \mathcal{R}_2 \rightarrow \mathcal{R}_3 \rightarrow$ Server, as shown in Figure 2. The attacker is located at $\mathcal{R}_2$ and is capable of eavesdropping the communications along

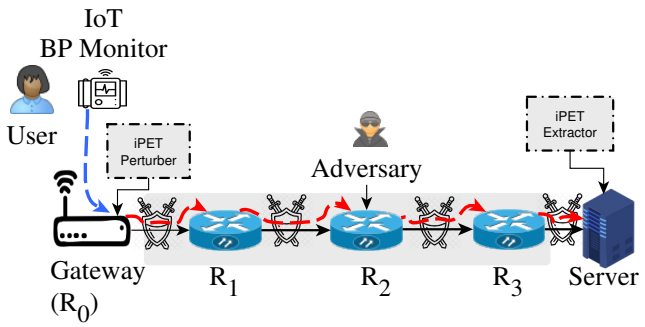


Figure 2: iPET, the proposed traffic perturbation technique employed at the periphery (i.e., $\mathcal{R}_0$ and Server) of the network that is under threat from the attacker.

the network path $\mathcal{R}_0 \rightarrow$ Server, to perform device fingerprinting. The goal of the IoT user (who uses a defense solution) is to effectively counter the ML-based fingerprinting attacks carried out by the network attacker. This means, the defense should be able to significantly reduce the probability of successful device classification from the currently achievable very high accuracy (of over 99%, as per our experiments in Section 5.1).

As mentioned in Section 2.2, an attacker trains a fingerprinting model by extracting information from network traffic; such a model learns specific characteristics of traffic (even if the payloads are encrypted), to construct decision boundaries that differentiate the patterns corresponding to different devices. To prevent this, the defense employed by the user should be able to obfuscate traffic by adding noise. For instance, the solution could add noise in the form of cover traffic (i.e., dummy packets with arbitrary but realistic sizes) to counter a fingerprinting model that uses packet count and size as features.

Challenges: The challenge in coming up with a traffic obfuscation technique is that it must be resilient to attackers that are aware of the defense solution. In particular, an attacker that learns that a specific defense solution is in place, may retrain his model accordingly to defeat the defense solution. In Appendix B, we show that Tor [11], a popular privacy enhancing overlay network, is easily defeated with our sequence-based fingerprinting model $\mathcal{M}_{\text{seq}}$ trained on Tor output ($\mathcal{M}_{\text{seq}}$ achieves $\approx 81\%$ accuracy). Furthermore, we need to ensure that the defense does not disrupt the functioning of the IoT devices by inducing high network latency or bandwidth overhead.

3.2 Overview of iPET

iPET is a deep learning-based traffic perturbation technique, and consists of a Perturber and an Extractor. iPET Perturber, which adds traffic perturbations, is deployed at the ingress of the network path that is vulnerable to fingerprinting attacks. Likewise, complementary to this, the iPET Extractor removes the perturbations at the egress of the vulnerable network path. Perturbations could be in the form of (i) cover traffic to be added (e.g., dummy packets), (ii) additional bytes to be added to the packets, or (iii) small and limited delays to be added to the packets from the IoT devices.

For illustration, we assume a simplified deployment model where iPET Perturber is deployed at the home Gateway ($\mathcal{R}_0$) and the iPET Extractor is deployed at the Server as shown in Figure 2. Real-world deployments must consider the trust assumptions of the involved network nodes and, additionally, consider real-world cases that may require routing the traffic through multiple hops. We discuss these in Section 6.

We present the design of iPET progressively. iPET-Basic (Section 4.2) is a generative model that creates perturbation vectors, which when added to the original device traffic, causes the attacker's fingerprinting model to misclassify. This means that the fingerprinting model's inference of the make and model of the devices fails with a very high probability. However, this form of defense is still vulnerable against an attacker who retrains his device fingerprinting model by deploying iPET at their end (refer to Section 4.2.3). To thwart such *retraining attacks*, we present an enhanced defence iPET-Advanced (Section 4.3), designed to enhance stochasticity between trained instances of the defense. The stochasticity ensures differences in the perturbations added to the traffic in different instances deployed by users. Thereby, an attacker's fingerprinting model trained on traffic perturbed by an iPET-Advanced instance they have access to, will not generalize on the traffic as perturbed by a specific victim's iPET-Advanced. We later go a step further and show that iPET-Advanced is resilient against an even more powerful attacker (discussed in Section 4.1) that may retrain on multiple such iPET-Advanced defense models to account for the stochasticity or even learn the defense parameters and architecture.

4 IPET

iPET uses a deep learning model to generate traffic perturbations, which when added to device traffic, will likely lead to misclassification of the attacker's ML classifier used for device fingerprinting. The input to iPET is a one-dimensional vector $\xi \in [0,1]^k$ ($k = 20$) drawn from a Uniform distribution, and the output is a one-dimensional perturbation vector δ of length n; Figure 3 illustrates this. The i^{th} element of this vector specifies a perturbation value, which determines how the traffic should be perturbed at a discrete slot ω_i, where $1 \leq i \leq n$. The values in the perturbation vector are used by iPET for a session of length $n \times |\omega|$. The perturbation value for a time-slot stipulates the number and aggregate size of dummy packets to be added. We do not explore the use of other forms of perturbations, such as packet delays, since it is easily influenced by network conditions and requires complex buffering mechanisms. However, the design of our defense is flexible and can be easily extended to include other forms of traffic perturbations.

iPET trains on the traffic of IoT devices to generate perturbations that will obfuscate potentially discernible patterns over sessions of fixed length ($n \times |\omega|$).

We introduce the attack scenarios in Section 4.1. We describe the foundation of iPET design, namely, iPET-Basic in Section 4.2. Subsequently, we develop iPET-Advanced, which is the final model referred to as iPET.

4.1 Attack Scenarios

i) Naïve Attacker ($\mathcal{M}_{\text{seq}}^{\text{naive}}$). A Naïve attacker deploys his own IoT devices, captures their (possibly) encrypted traffic at his IoT gateway, and labels them. He uses this labeled dataset for training a sequence-based classifier $\mathcal{M}_{\text{seq}}^{\text{naive}}$ (same as $\mathcal{M}_{\text{seq}}$, defined in Section 2.2). As discussed in Section 2.2, the sequence-based classifiers perform better than aggregate ones.

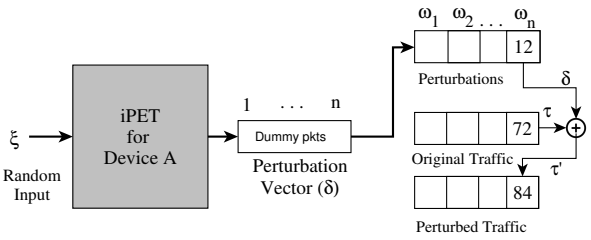


Figure 3: The input to iPET is a random vector ξ, and the output is a perturbation vector δ that specifies, for a specific time-slot ω_i, the number of dummy packets along with the packet sizes to be added.

ii) Retraining Attacker ($\mathcal{M}_{\text{seq}}^{\text{retrained}}$). Different from the Naïve attacker, a Retraining attacker deploys iPET on an IoT gateway, gaining blackbox access to the iPET gateway. In other words, the attacker deploys a set of devices, captures perturbed traffic from the egress of iPET gateway, and retrains his fingerprinting classifier. The attacker uses this iPET-aware ML classifier on (victim) user traffic for device fingerprinting. We denote this retrained sequence-based classifier as $\mathcal{M}_{\text{seq}}^{\text{retrained}}$.

iii) Powerful Attacker ($\mathcal{M}_{\text{seq}}^{\text{powerful}}$). A Powerful attacker operates *several* iPET gateways. He can now deploy IoT devices at each of his iPET gateways, capture their traffic, and label them. The attacker builds a more robust ML classifier that is trained on the traffic traces from several iPET instances running at the gateway he controls. The new ML classifier thus trained is called $\mathcal{M}_{\text{seq}}^{\text{powerful}}$. Additionally, we also assume that the attacker has learned the model parameters and hyper parameters of iPET model (Section 4.3) from the several instances that he has access to. However, he does not have access to the exact weights of the neural network models trained at the iPET instances for the different users.

4.2 iPET-Basic

The design of iPET-Basic is inspired by Generative Adversarial Perturbations [36], which utilizes a generative network (i.e., a generator) to formulate perturbations for defeating an image classifier (i.e., a discriminator) with high confidence. For instance, when a perturbation from the generator is added to an image of a dog, the discriminator will label it as a cat with a high probability. Analogous to this, in our work, iPET-Basic is the generator ($\mathcal{G}$) and the discriminator ($\mathcal{D}$) is the naïve attacker ($\mathcal{M}_{\text{seq}}^{\text{naive}}$). The basic idea is that $\mathcal{G}$ progressively learns to generate better perturbations that defeat $\mathcal{D}$.

4.2.1 Model Design. **Model Architecture.** The generator's model architecture is described in Fig 4. We utilize a dense layer, where each neuron learns an element in the perturbation vector δ. For example, the number of packets to be added in the discrete slot is learned by a neuron. The dense layer output is reshaped into the perturbation vector δ. The custom limit layer is included to scale the perturbation values within the constraints imposed either

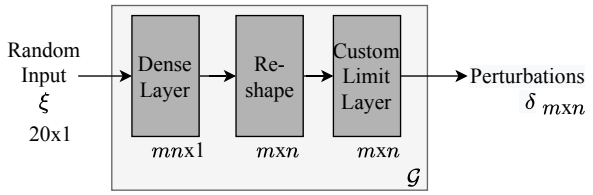


Figure 4: Model architecture of the generator ($\mathcal{G}$).

by the user (e.g., for saving bandwidth) or the network protocol (e.g., the maximum allowed payload size). The following are some relevant constraints:

- Control the number of dummy packets to add, and their sizes, per discrete slot (ω_i).
- Ensure that the payload bytes for the dummy packets do not result in the packet size exceeding the MTU (Maximum Transmission Unit) imposed by the network.
- Ensure that the delay (in milliseconds) added to original packets is within the user specified limits

We define control parameters for the above, which help iPET to achieve a trade-off between privacy and bandwidth overhead, as we demonstrate later in Section 5.1.2. In this work, the custom limit layer controls only the number and size of the dummy packets since we choose not to include the other forms of traffic perturbation. However, as presented above, the layer can additionally control other perturbation types as needed.

Training. The objective of iPET-Basic is to generate perturbations $\mathcal{G}(\xi) = \delta$, which when added to an input time series τ, creates an adversarial input $\tilde{\tau} = \tau \oplus \delta$, that causes the discriminator $\mathcal{D}$ to misclassify the original traffic session τ. An example of the working of $\oplus$ operator is shown in Figure 3. The perturbed traffic, along with the silence period s of the device, is now used as an input to the discriminator, as summarised in Figure 5. The discriminator predicts the device type for the given input. Finally, the prediction is used as a feedback to update the generator weights, and this process continues.

There are two well-known approaches to generate perturbations. i) *Targeted approach* [36, 39]: the generator attempts to force the discriminator to misclassify the input into a specific target class. ii) *Untargeted approach* [32, 36]: the generator does not have a specific goal of forcing a misclassification into a particular class label. Instead, the generator receives a positive feedback as long as the discriminator misclassifies the input. The open-ended nature of the training objective could likely result in a stochastically different iPET generator being created each time the model is trained. This is mainly due to the freedom in misclassifying an input into any of the available classes. We, therefore, choose the untargeted approach for our work. Thus, we train our generator $\mathcal{G}$ such that,

$$\mathcal{D}(\tilde{\tau}, s) \neq c_{\tau,s}\,, \tag{1}$$

where τ is a traffic session (input time series of feature values), s is the silence period of the IoT device, and $c_{\tau,s}$ is the ground truth label.

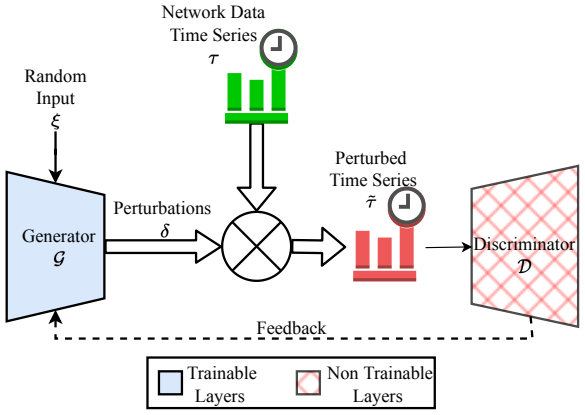


Figure 5: Overview of generator training.

Loss Function. The loss function can thus be defined as a decreasing function of categorical cross-entropy, denoted by

$$l = \sum_{i=1}^{C} y_i \ \log(\hat{y}_i), \tag{2}$$

where, $\hat{y}$ is the model output and y is the corresponding ground truth label, both of which are one-hot encoded. Following common practice, the model is trained for sufficient number of epochs to learn the perturbations that can deceive the discriminator with high confidence.

4.2.2 Stochastic Differences between Generator Instances. In iPET, every user gets a different instance of iPET-Basic installed at their iPET gateway. The different instances (e.g., $\mathcal{G}^1, \mathcal{G}^2, \ldots, \mathcal{G}^{\#users}$) of iPET-Basic have the following similarities. i) The discriminator $\mathcal{D}$ (refer Figure 5), which is used for training the different iPET-Basic generator instances, is the same. ii) The model architecture and the hyper-parameters of the generator instances are also the same. However, due to their initial random weights and the untargeted training objective, the final weights of the generators are likely to be different. Thus, effectively, the perturbations generated by the instances at a user ($\mathcal{G}^{user}$) and an attacker ($\mathcal{G}^{adv}$) are different; this is illustrated in Figure 6(a).

4.2.3 Attack against iPET-Basic.

Naïve Attacker. In this scenario, the attacker uses the discriminator $\overline{\mathcal{D}} = \mathcal{M}_{\text{seq}}^{\text{naive}}$ to carry out fingerprinting attack against a user that employs iPET-Basic generator at her premises.

Retraining Attacker. Figure 6(a) depicts generator instances at the user ($\mathcal{G}^{user}$) and the attacker ($\mathcal{G}^{adv}$). Subsequently, as show in Figure 6(b), the attacker, at his end, retrains a new discriminator ($\overline{\mathcal{D}} = \mathcal{M}_{\text{seq}}^{\text{retrained}}$) using the instance of iPET-Basic generator ($\mathcal{G}^{adv}$) available on his gateway, with an intention of fingerprinting devices deployed by the user.

As we will see later in Section 5.3.1, $\mathcal{M}_{\text{seq}}^{\text{retrained}}$ can easily fingerprint devices even after traffic is perturbed by iPET-Basic, thereby motivating the need for a better defense.

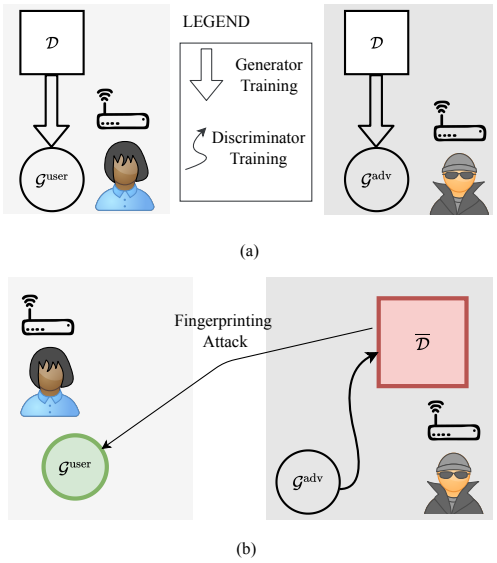


Figure 6: (a) Two instances of iPET-Basic generators, one at a user and another at an attacker. (b) The Retraining attacker builds a stronger discriminator $\overline{\mathcal{D}}$ based on $\mathcal{G}^{adv}$ at his own iPET gateway.

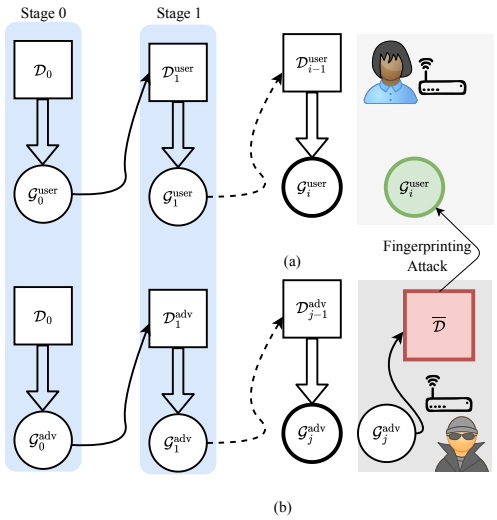


Figure 7: (a) The training process for creating generator instance $\mathcal{G}_i^{user}$, and its deployment at a user. (b) The training process for creating a generator instance $\mathcal{G}_j^{adv}$ at an attacker, from which he builds his discriminator $\overline{\mathcal{D}}$.

4.3 iPET-Advanced

We first present the design of iPET-Advanced followed by a discussion on the attack scenarios against iPET-Advanced.

4.3.1 Model Design. iPET-Advanced is build with iPET-Basic's generator architecture (refer Section 4.2) as the fundamental building block. Following the motivation shared in the Section 4.2, the generator uses an untargeted training objective. The goal of iPET-Advanced is to ensure that the perturbations of the generator instances at the gateways differ *significantly* from one another. Intuitively, the idea is to make it challenging for an attacker to use the decision boundaries that he learns from his own generator instance for employing against the traffic captured at another iPET user.

As with iPET-Basic, we use a GAN-based architecture for iPET-Advanced. The distinct and important design principle of iPET-Advanced is to chain a series of discriminator-generator training sequences such that, at each stage, the perturbation patterns learned by the generator diverges, allowing more random values to be explored in the state space. For illustration, let us take the example of a generator instance trained specifically for a user, shown in Figure 7(a). The subscript of the model denotes the *stage* in the chain and the superscript denotes the user for which the model is created. We begin with the discriminator $\mathcal{D}_0$ (same as the $\mathcal{D}$ used in Section 4.2) on which the generator ($\mathcal{G}_0^{user}$) is trained. Subsequently, the $\mathcal{D}_1^{user}$ is trained to fingerprint the traffic as perturbed by $\mathcal{G}_0^{user}$. This newly trained discriminator is now used to create the second stage generator ($\mathcal{G}_1^{user}$). Although the magnitude of perturbations added at any stage remains small, the untargeted training objective moves the data points randomly. The key idea of this process is that the stochasticity introduced by the untargeted training objective is magnified by the iterative training process since the generator at each stage is trained using a different discriminator as a starting point. Therefore, the state space for the perturbations is vast, resulting in sharply diverging generators between two iPET-Advanced instances.

Observe that from Stage 1 onward, the discriminator on which the user and the attacker train (i.e., $\mathcal{D}_1^{user}$ and $\mathcal{D}_1^{adv}$) are different, unlike Stage 0 (equivalent to iPET-Basic) where both train their generators on the same $\mathcal{D}_0$. Therefore, a generator $\mathcal{G}_i^{user}\ \forall\ i \geq 1$ will be suitable for use as the iPET-Advanced generator. Figure 7(a) shows a chain, where $\mathcal{G}_i^{user}$ is chosen for installation at a user's iPET gateway. Note that only the generator at the end of the chain is installed at the iPET gateway and the intermediate models are not available to the iPET users. This design also ensures that the generator installed at different users is likely to be different.

4.3.2 Attacks against iPET-Advanced.

Naïve Attacker. In this straight-forward attack scenario, the attacker uses the discriminator $\overline{\mathcal{D}} = \mathcal{D}_0$ to contend against the users' generator $\mathcal{G}_i^{user}$.

Retraining Attacker. Figure 7(b) shows the chain of discriminator-generator sequences that leads to the generator instance $\mathcal{G}_j^{adv}$ installed at the attacker's iPET gateway. The attacker uses this generator instance to retrain his ML classifier, namely, $\overline{\mathcal{D}} = \mathcal{M}_{\text{seq}}^{\text{retrained}}$.

Powerful Attacker. We now describe the attack scenario where the attacker (refer Section 4.1) creates a powerful discriminator $\overline{\mathcal{D}} = \mathcal{M}_{\text{seq}}^{\text{powerful}}$ to contend against the users' generator $\mathcal{G}_i^{user}$. Figure 8(a) shows the access that a Powerful attacker has over the several iPET gateways he controls. When the attacker uses the iPET-Advanced at the gateways as a blackbox, he will have access to the several last-stage generators such as $\mathcal{G}_j^{adv}$ and $\mathcal{G}_k^{adv}$ shown

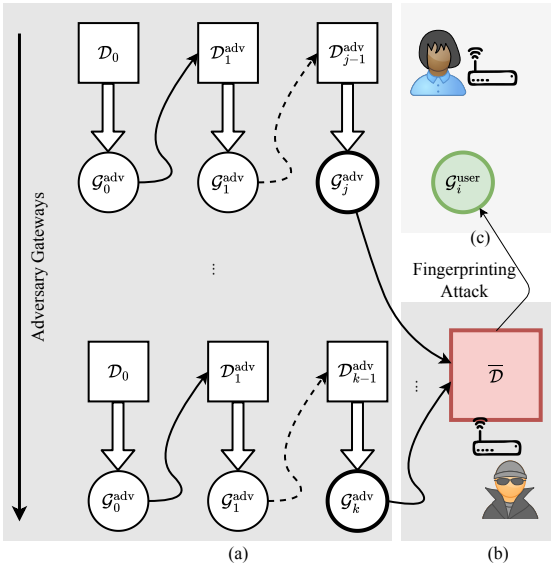


Figure 8: (a) Different generators that the attacker can build based on its access to several iPET gateways. When the attacker learns the iPET-Advanced architecture, he will also have access to intermediate generators shown here. (b) The attacker builds a discriminator $\overline{\mathcal{D}}$ based on the generators available to him (c) The attacker attempts to fingerprint a user who uses a iPET-Advanced based generator $\mathcal{G}_i^{user}$.

in the Figure. However, if he learns the architecture and parameters of iPET-Advanced, he can even recreate the intermediate stage generators. Figure 8(b) shows the attacker training his $\overline{\mathcal{D}} = \mathcal{M}_{\text{seq}}^{\text{powerful}}$ based on the generators that he has access to. In Figure 8(c), the attacker uses $\overline{\mathcal{D}}$ to fingerprint the iPET-Advanced based generator at the user. This attack scenario is evaluated in detail in Section 5.3.3. To the best of our knowledge, no previous research works in the IoT domain have evaluated their defenses against an attacker as powerful as $\mathcal{M}_{\text{seq}}^{\text{powerful}}$.

5 PERFORMANCE EVALUATIONS

Implementation. We implement iPET and the fingerprinting models in Python, using Keras libraries from TensorFlow $v2.5.0$ [52]. iPET generates a perturbation vector (δ) intended to be used for a session, such that the session length is $n \times |\omega| = 4$ seconds, where $|\omega| = 0.1$s and $n = 40$. The perturbation vector is added to the original device traffic, to prevent device fingerprinting attack. Generated perturbations are then added to the original device traffic using a python program.

Real-World Dataset. We conduct our experiments on real world IoT device traffic from the UNSW Smart Home Traffic Dataset [47]. The dataset consists of traffic traces collected from 30 devices over 20 days, amounting to 20 pcap files, each representing a day's worth of data. We set our vantage point to be the IoT gateway, with the devices communicating via the gateway to servers on the Internet. For our evaluations, we do not consider traffic that is internal to the home network that our attacker cannot observe.

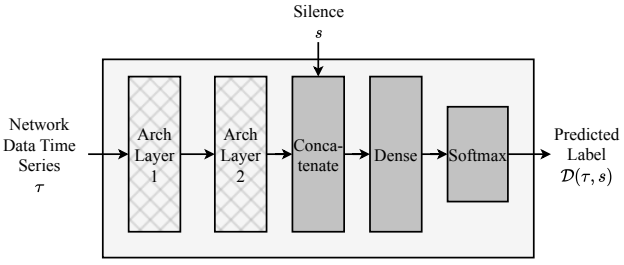


Figure 9: Generalised Architecture for $\mathcal{M}_{\text{seq}}$, taking a time series τ and the silence duration s as input, to predict the corresponding device.

Evaluation Metrics. We use weighted F1-score of device fingerprinting accuracy to measure the effectiveness of the defense. Additionally, bandwidth overhead of the defense is measured in *KB/s*. The objective of the defense is to minimize fingerprinting accuracy of attacker while ensuring overheads are minimal.

Evaluation Scenarios. We attempt to answer several research questions (**RQ**s) as part of performance evaluation. In Section 5.1, we measure the effectiveness of iPET against several device fingerprinting models and compare it with defenses from the literature. In Section 5.2, we show the resilience of iPET when the attacker exploits time-series parameters to come up with a better fingerprinting model. Subsequently, in Section 5.3, we provide further insights into the design of iPET by comparing the efficacy of iPET-Basic and iPET-Advanced against attackers with varying degrees of capabilities (introduced in Section 4.1). Finally, in Section 5.4, we evaluate iPET in an *open-world* scenario.

5.1 Performance of iPET

5.1.1 *RQ1: Is iPET Resilient to State-of-the-Art Fingerprinting Techniques?*

Attack Setup. We consider the fingerprinting model $\mathcal{M}_{\text{seq}}^{\text{retrained}}$ (described in Section 2.2). The following deep learning architectures are considered in our evaluations: Bi-LSTM [17, 42], CNN [21], GRU [7], and MLP [40]. The generic architecture for $\mathcal{M}_{\text{seq}}$ is depicted in Figure 9. We use the notation $\mathcal{M}_{\text{seq:Bi-LSTM}}^{\text{retrained}}$ to denote that the $\mathcal{M}_{\text{seq}}^{\text{retrained}}$ model uses Bi-LSTM units to replace Layer 1 and Layer 2 in Figure 9.

Defense Setup. iPET requires training a generator model that can be used to generate perturbations. For this experiment, we use four different iPET generators, namely, $\text{iPET}_{\text{Bi-LSTM}}$, iPET_{CNN}, iPET_{GRU},and iPET_{MLP}, which use the discriminator, namely, Bi-LSTM, CNN, GRU, and MLP, respectively. In a real-world setting, the best performing generator architecture can be chosen by the defense. Refer to Figure 4 for details of the generator architecture.

Result: We consider a realistic scenario where the attacker and defense are allowed to choose a classifier whose details are not known to the other. Table 1 shows the fingerprinting accuracy (weighted F1-Score) of various attacker fingerprinting models against various iPET defense models. Each value in the table (e.g., $\mathcal{M}_{\text{seq:CNN}}^{\text{retrained}}$ attack

iPET-Defenses (*iPET*)	Attacker Models for $\mathcal{M}_{\text{seq}}$			
	Bi-LSTM	**CNN**	**GRU**	**MLP**
Undefended	99.75	96.36	99.5	99.42
Bi-LSTM	6.76	7.07	4.54	1.34
CNN	14.55	8.65	8.59	2.82
GRU	4.93	3.55	4.08	1.08
MLP	25.47	7.98	21.81	2.97

Table 1: Weighted F1-score achieved by various attackers against iPET trained with different DL architectures.

model against $\text{iPET}_{\text{Bi-LSTM}}$) is an average of 18 experiments. For reference, the first row of the table shows the case when the traffic is undefended, where, unsurprisingly, all attack models identify the devices with near perfect F1-score. Observe that, even in the worst case — iPET_{MLP} defense against $\mathcal{M}^{\text{retrained}}_{\text{seq:Bi-LSTM}}$ — the F1-score is limited to $\approx$ 25%, showing the effectiveness of iPET. Overall, iPET_{GRU} and $\text{iPET}_{\text{Bi-LSTM}}$ are the most effective defenses against all considered attack models. Most importantly, we find that the attacker's choice of training architecture does not provide undue advantage when iPET is used as a defense.

5.1.2 *RQ2: How Does iPET Compare to Other Traffic Obfuscation Techniques?*

We now compare iPET with traffic obfuscation techniques from the literature. Following are the details of the traffic obfuscation techniques used for comparison.

Constant Rate [1, 3]. We implement a version of this defense based on Stochastic Traffic Padding (STP [1]). In this technique, when a device starts transmitting at time t, we begin injecting cover traffic in the form of dummy packets to maintain the constant bit-rate of $\mathcal{R}$ for a time period T. If there is no device activity, at the beginning of every interval T, a *decision function* decides if cover traffic must be added. The decision function is defined as a simple Bernoulli trial that succeeds with probability Q. The defense will always add cover traffic when $Q = 1$, and cover traffic will only be added when the device is transmitting if $Q = 0$.

Random Rate. Introducing randomness in the traffic traces can intuitively defeat device fingerprinting models. We develop a traffic obfuscation defense where a target bit-rate $\mathcal{R}_{rand}$ is chosen randomly from a normal distribution $\mathcal{N}(\mu, \sigma)$. This bit-rate is maintained by the addition of dummy packets or by padding packets until it reaches $\mathcal{R}_{rand}$. Further, $\mathcal{R}_{rand}$ is re-sampled once every T_{rand} seconds for more randomness. We reduce potentially high bandwidth costs by introducing a probabilistic decision function similar to the Constant Rate technique.

Random Rate+. To further increase stochasticity, we develop an advanced version of the Random Rate technique where the mean of the normal distribution itself is randomized for every device. Specifically, a mean μ_r is drawn from a uniform random sample $\mathcal{U}_{[\mathcal{R}_{min}, \mathcal{R}_{max}]}$ and assigned to each device. A target bit-rate $\mathcal{R}_{rand}$ is then drawn from the normal distribution $\mathcal{N}(\mu_r, \sigma)$ and applied every T_{rand} seconds. A probabilistic decision function reduces the bandwidth overhead.

MTU Padding [55]. Every network packet is padded to the network's Maximum Transmission Unit (MTU) size.

Tor. Although not known for its traffic obfuscation, Tor is a widely used practical privacy system that implements a traffic obfuscation algorithm. Tor uses fixed-sized cells (512 bytes) and background noise [38] when transmitting network packets. Our intention is to see if this is sufficient in providing some level of protection against device fingerprinting attacks. To implement this, we used a Tor emulation framework—Shadow [19]—and the official Tor release v0.4.4.8 [37].

iPET. We use $\text{iPET}_{\text{Bi-LSTM}}$ described in Section 5.1.1 for this experiment and for all experiments henceforth. To control the bandwidth overhead, iPET can be configured with parameters [max no. of dummy packets, max payload size of dummy packets] which limit the maximum bandwidth required in a given time-slot ω_i (Section 4.2.1).

Defenses	Attacks ($\mathcal{M}_{\text{seq}}$)	
	Bi-LSTM	**CNN**
Tor	81.6	87.1
Constant Rate ($\mathcal{R}$=10, Q=0)	95.28	88.9
Constant Rate ($\mathcal{R}$=10, Q=0.45)	54.26	33.76
Constant Rate ($\mathcal{R}$=10, Q=1)	56.47	30.09
Random Rate ($\mathcal{R}_{rand}$~$\mathcal{N}$(18.5,3.6), Q=0.5)	56.71	41.61
Random Rate+ (μ_r~$\mathcal{U}_{[5,40]}$, $\mathcal{N}(\mu_r, 10)$, Q=0.5)	45.84	25.35
MTU Padding	92.4	83.8
iPET	**14.72**	**17.74**

Table 2: Weighted F1-score achieved by attackers $\mathcal{M}^{\text{retrained}}_{\text{seq:CNN}}$ and $\mathcal{M}^{\text{retrained}}_{\text{seq:Bi-LSTM}}$ on IoT devices protected by different defenses. All bit-rate parameters are represented as KB/s.

Attack Setup. We use $\mathcal{M}^{\text{retrained}}_{\text{seq:CNN}}$ and $\mathcal{M}^{\text{retrained}}_{\text{seq:Bi-LSTM}}$ fingerprinting models for the attacker in these experiments. As observed in Table 1, these two attack models perform the best when $\text{iPET}_{\text{Bi-LSTM}}$ is deployed as defense.

Results Table 2 summarizes the results. Our results show that iPET has a very low F1-score against both the attackers, effectively thwarting the fingerprinting attack. Tor and the MTU Padding defense perform poorly (F1-score > 80%), since the attacker uses a sequence-based classification model for fingerprinting. With MTU-padding, the number of incoming and outgoing packets remains unperturbed, allowing the attacker to model the traffic rate effectively. For the constant rate technique, we evaluate three different configurations. The time period parameter T is set to 1 second in each configuration but we vary the rate $\mathcal{R}$ and probability Q. For $\mathcal{R}$ = 10KB/s and Q = 0, the defense performs poorly, as sporadic device activity may be discernible when the attacker uses a sequence-based model. Even at higher values of Q, the defense is able to reduce the accuracy of the attack to just over 50% (and ≈30%) against the $\mathcal{M}^{\text{retrained}}_{\text{seq:Bi-LSTM}}$ (and $\mathcal{M}^{\text{retrained}}_{\text{seq:Bi-CNN}}$) fingerprinting models.

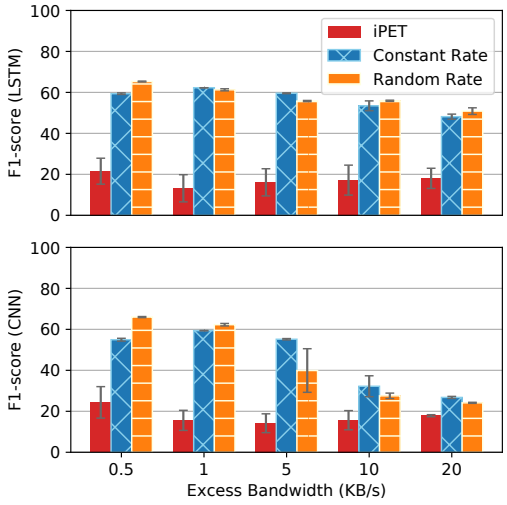


Figure 10: Effect of higher bandwidth on F1-score for attackers $\mathcal{M}^{\text{retrained}}_{\text{seq:CNN}}$ and $\mathcal{M}^{\text{retrained}}_{\text{seq:Bi-LSTM}}$.

The Random Rate defense configured with $\mathcal{R}_{rand}{\sim}\mathcal{N}(18.5, 3.6)$ reduces the accuracy to ≈50% and ≈41% against the $\mathcal{M}^{\text{retrained}}_{\text{seq:Bi-LSTM}}$ and $\mathcal{M}^{\text{retrained}}_{\text{seq:CNN}}$ attackers respectively.

On the other hand, the Random Rate+ defense configured with $\mu_r{\sim}\mathcal{U}_{[5,40]}, \mathcal{N}(\mu_r, 10), Q = 0.5$ performs much better against the $\mathcal{M}^{\text{retrained}}_{\text{seq:CNN}}$ attacker by limiting the F1-score to just ≈25%. However, when compared to the $\mathcal{M}^{\text{retrained}}_{\text{seq:Bi-LSTM}}$ attacker, the accuracy is relatively higher at ≈45%. Yet, it may be possible to improve the Random Rate+ defense by experimenting with different combinations of distributions and their parameters. However, this would require large number of experiments, as the state space that needs to be explored to find the optimal configuration is vast (per-device distributions, their parameters, etc. have to be estimated). Further, the choice of parameters must also account for costs in terms of bandwidth and latency. This can be viewed as an optimization problem. On the contrary, iPET utilizes the strong function approximation characteristics of neural networks to automate this process.

Traffic obfuscation techniques are known to perform better when there are no limits on the bandwidth they consume. Figure 10 plots the performances of the best performing traffic obfuscation techniques that allow us to control the bandwidth overhead. We can see that the performance of both constant rate and random rate techniques improve at a higher bandwidth overhead (e.g., 20 KB/s). iPET, however, performs better even at a significantly lower bandwidth overhead of 0.5 KB/s.

5.1.3 *RQ3: What is the Effect of Fingerprinting Attack on Different IoT Devices in the Presence of iPET Defense?*

We now analyze how iPET defends different IoT devices. Figure 11 presents a fine-grained analysis of the defenses using a CDF of the F1-score measured per device. We can see that, for iPET, about 50% of the devices have an F1-score less than 20 and no device has a F1-score greater than 45.

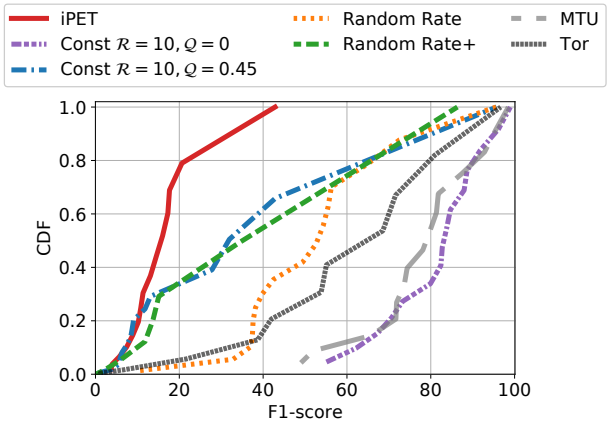


Figure 11: CDF of F1-score of IoT devices for attackers $\mathcal{M}^{\text{retrained}}_{\text{seq:CNN}}$.

Furthermore, we group the IoT devices in our dataset into five categories based on their use case, and compare iPET and the Constant Rate defense. We choose this defense since it allows us to control the bandwidth overhead to be roughly the same, so that a fair comparison with iPET can be made. Figure 12a compares the F1-score of the IoT devices within popular categories based on their use-case. We observe that iPET outperforms in all the categories. The fingerprinting attack does not achieve more than ≈ 20% against iPET. Figure 12b provides further insights on the worst performing categories, namely, AI-Assistants and Smart Cameras.

5.2 Exploiting Time-Series Parameters

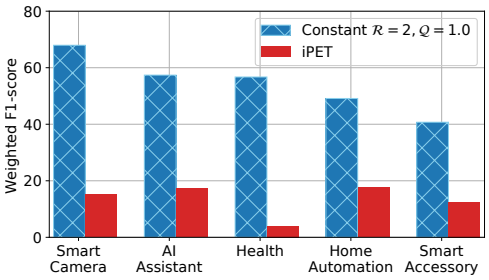
In this section, we evaluate the case where the attacker uses a fingerprinting model that uses a time-series parameter that is different from that of the discriminator iPET uses to train against, with an intention of coming up with a better fingerprinting model. All results reported in this section are average over 18 test runs.

Defense and Attack Setup. We use the $\text{iPET}_{\text{Bi-LSTM}}$ as the defense model, and $\mathcal{M}^{\text{retrained}}_{\text{seq:CNN}}$ as the attack model for the evaluations presented in this section.

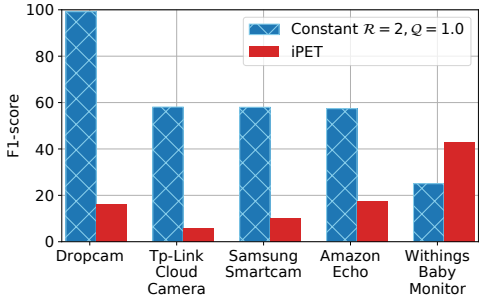
5.2.1 *RQ4: Is iPET Effective Against an Attacker Who Uses a Different Input Time-Series Parameters in Their ML Classifiers?*

Perturbation Slot Length $|\omega|$. Perturbation slot length, which is used for calculating the features within the time series input τ is an important feature for a fingerprinting model. In this experiment, we vary the value of $|\omega|$ but use a fixed observation time of 4 seconds to see if the attacker's fingerprinting model can discern useful patterns. The defense model considered remains unchanged, generating perturbations for $|\omega| = 0.1$s. Table 3 shows that the attacker does not achieve any substantial improvement in the F1-score by changing $|\omega|$.

Observation Time $n \times |\omega|$. This is an important parameter of time-series classification models used by both the attacker as well as the



(a) iPET vs. Constant Rate, for different IoT device categories against $\mathcal{M}^{\text{retrained}}_{\text{seq:CNN}}$



(b) iPET vs. Constant Rate, for AI-Assistants and Smart Cameras against $\mathcal{M}^{\text{retrained}}_{\text{seq:CNN}}$

Figure 12: Performance of iPET on different IoT devices.

iPET generative model. In Figure 13, the X-axis plots the various observation time used by the attacker's fingerprinting model. Each line plotted in the graph represents an iPET instance that uses a different observation time for modeling perturbations. The Y-axis plots the F1-score achieved by the attacker against the respective model used by iPET. When iPET uses an observation time of 4 seconds, the attacker achieves a maximum F1-score of $\approx 80\%$ with an observation time of 0.5 seconds and the F1-score reduces with increase in attacker's observation time. However, we can observe that the attacker's fingerprinting model is worse when iPET also uses smaller observation times (e.g., 0.5s). Therefore, we recommend that the observation time chosen by an iPET instance be as small as possible.

$\lvert\omega\rvert$ (s)	0.05	0.1	0.2	0.5	1
$\mathcal{M}^{\text{retrained}}_{\text{seq:Bi-LSTM}}$	27.44	26.27	29.28	24.18	28.96
$\mathcal{M}^{\text{retrained}}_{\text{seq:CNN}}$	16.18	14.72	18.58	23.05	23.14

Table 3: Weighted F1-score achieved by attacker on Varying $\lvert\omega\rvert$ in captured Time Series. The defence is fixed, adding perturbations for each $\lvert\omega\rvert = 0.1s$.

5.3 Evaluating iPET-Basic and iPET-Advanced

Synthetic Dataset. In addition to the real-word dataset, in this section, we consider a synthetic dataset for our evaluations. Our

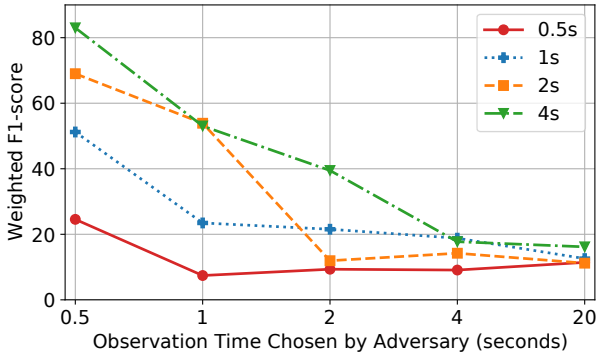


Figure 13: Varying the total observation time for the Time Series as modeled by the attacker and Defense. The sampling rate ω remains constant.

synthetic dataset consists of data points from 50 classes (devices), each with 50,000 samples. The points are sampled from Gaussian Mixture Model (GMM), one for each class. GMMs are commonly used for modelling real-world network traffic (e.g., for traffic classification [12, 61]). The GMM itself is composed of five Gaussian distributions. We also added sufficient noise to the data and ensure minor overlaps between classes to emulate real-world behavior. For reference, a $\mathcal{M}^{\text{naive}}_{\text{seq:CNN}}$(Section 4.1) classifier achieves an F1-score of 98% in fingerprinting the synthetic classes when undefended.

Defense and Attack Setup. We compare the effectiveness of iPET-Basic and iPET-Advanced against the attack scenarios that we described in Section 4.1. As before, we use the $\text{iPET}_{\text{Bi-LSTM}}$ as the defense model for the evaluations presented in this section. The attackers considered have the following characteristics:

i) $\mathcal{M}^{\text{naive}}$: An $\mathcal{M}_{\text{seq:CNN}}$ classifier trained on IoT device traffic and does not have knowledge of iPET generators.

ii) $\mathcal{M}^{\text{retrained}}$: An $\mathcal{M}_{\text{seq:CNN}}$ classifier trained on *one* iPET generator instance.

iii) $\mathcal{M}^{\text{powerful}}$: An $\mathcal{M}_{\text{seq:CNN}}$ classifier trained on *multiple* iPET instances

Evaluation Setup (Real World). The iPET generator models provided to real users as well as attackers (pretending to be users) are generated from iPET trained until Stage i, where $0 \leq i \leq 4$. This means that the generator instance available to them could be from any of the stages 0 to 4. While the $\mathcal{M}^{\text{retrained}}$ attacker has access to one iPET generator instance, the $\mathcal{M}^{\text{powerful}}$ is assumed to have access to 20 iPET generators. For statistical significance, we repeat the experiments several times with different user-attacker pair.

Evaluation Setup (Synthetic). The iPET generator models provided to real users as well as attackers (pretending to be users) are generated from iPET trained until Stage i, where $0 \leq i \leq 10$. In this case, $\mathcal{M}^{\text{powerful}}$ is assumed to have access to 44 iPET generators. We repeat these experiments several times for statistical significance.

5.3.1 ***RQ5: How Do iPET-Basic and iPET-Advanced Perform Against Different Attack Scenarios?*** Figure 14a plots the effectiveness of $\mathcal{M}^{\text{naive}}$ and $\mathcal{M}^{\text{retrained}}$ against iPET-Basic and iPET-Advanced. The Naïve attacker $\mathcal{M}^{\text{naive}}$ is unsuccessful in fingerprinting the devices protected by both iPET-Basic and iPET-Advanced. Whereas, the Retraining attacker $\mathcal{M}^{\text{retrained}}$ achieves a high F1-score of 89% against iPET-Basic. In comparison, iPET-Advanced is highly effective in countering fingerprinting attack by $\mathcal{M}^{\text{retrained}}$. We conclude that chaining the discriminator-generator pair in iPET-Advanced helps improve stochasticity of its generator, making it effective against a retraining attacker.

5.3.2 ***RQ6: How Does the Stage of the Generator Affect the Performance of iPET-Advanced?*** We analyze how the training stage affects the resilience of iPET-Advanced. Figure 14b and Figure 14c present the results for the real-world and synthetic data, respectively. The X-axis denotes the iPET generator $\mathcal{G}_i$ used to defend the user's traffic. Here i is the training stage of the iPET generator. Each line in the graphs represents a particular attack scenario. For example, the $\mathcal{M}^{\text{retrained}}$ attacker $\overline{\mathcal{D}}_j$ represents an attacker who has access to their own iPET generator from Stage j.

We observe that the attacker's discriminators do not achieve a weighted F1-score of more than $\approx$ 20% against iPET-Advanced generators, i.e., when the generator is at Stage $i \geq 1$.

5.3.3 ***RQ7: Is iPET-Advanced Resilient to the Powerful Attacker (Section 4.1)?*** Figure 14b shows that, on the real-world dataset, although $\mathcal{M}^{\text{powerful}}$ performs better than the Retraining attacker $\overline{\mathcal{D}_j} = \mathcal{M}^{\text{retrained}}$, the maximum F1-score against iPET-Advanced is $\approx$50%. On the synthetic dataset, we get even better results, with the F1-score of $\mathcal{M}^{\text{powerful}}$ converging to $\approx$20%, demonstrating the effectiveness of the perturbations generated by iPET-Advanced.

5.4 iPET in the Open World

Our evaluations so far have considered a closed-world scenario, where the attacker trains and tests their classifier on traffic samples from a *monitored* set of devices. In a realistic *open-world* scenario, the attacker not only trains on a small set of all devices, but must also distinguish if a sample belongs to the monitored device set or not. We evaluate the accuracy of the device fingerprinting attack in such an open-world setting. We consider $\text{iPET}_{\text{Bi-LSTM}}$ as the defense model, and $\mathcal{M}^{\text{retrained}}_{\text{seq:Bi-LSTM}}$ as the attack model for the evaluations presented in this section.

Datasets. We utilise the real-world dataset and the synthetic dataset introduced in Section 5.3.

5.4.1 ***RQ8: Is the Device Fingerprinting Attack Effective in an Open-World Scenario?***

Attack Setup. From the synthetic dataset containing 50 devices, we categorize devices 1–30 as monitored devices, and 31–50 as unmonitored ones. The unmonitored devices are labeled as a single unmonitored class in the training set. Devices 41–50 are not included in the training set. Therefore, we train $\mathcal{M}^{\text{retrained}}_{\text{seq}}$ with only 31 classes, with the objective to classify the 30 monitored devices correctly, and the remaining devices as unmonitored.

Results. In the open-world scenario with no defense employed (i.e., the undefended gateway), the $\mathcal{M}^{\text{retrained}}_{\text{seq}}$ classifier classifies the traffic samples with an F1-score of 87.39%. Although we note a drop in the fingerprinting accuracy from the closed-world, where we observe an F1-score of 98%, the accuracy is still high enough to motivate the need for a defense solution.

Defending the traffic against a fingerprinting attack with iPET, the fingerprinting accuracy drops to less than a random guess with an F1-score of 1.65%. Here, we have an iPET instance for each monitored device and one to generate perturbations for all the unmonitored devices.

5.4.2 ***RQ9: Can an Attacker Identify the Device Category in an Open-World Scenario?*** **Attack Setup.** As discussed in 5.1.3, we categorise the 30 devices in the real-world dataset into 5 categories based on their functionality, and the rest (e.g., Laptops, Mobile phones, tablets) into a new category called background devices. To emulate an open world scenario, we drop a few devices from each category, leaving 22 devices in the training set. An $\mathcal{M}^{\text{retrained}}_{\text{seq}}$ model is then trained to classify the network flow into one of the six categories.

Results. When no defense is used, the model effectively identifies the device category with an F1-score of 95.13% in the open-world scenario, compared to 99.53% in the closed world scenario. For iPET, we train one iPET instance for each device category instead of having one for each device. Therefore, the same instance will obfuscate the traffic for all the devices in a given category. The F1-score achieved by an $\mathcal{M}^{\text{retrained}}_{\text{seq}}$ adversary to identify the device category once the defense is deployed drops to an insignificant 19.93%.

6 DISCUSSIONS

Deployment Options. Deployment of iPET involves placing the iPET Logic (i.e., the iPET Perturber and iPET Extractor modules) at the ingress and egress of the network path that the user trusts. Implementing iPET Perturber and iPET Extractor at an IoT user's home gateway and the server, respectively, is an option; as shown in Figure 2. However, there is a dependency on servers to modify code to support iPET. Furthermore, several IoT devices can be easily mapped to their servers (i.e., destination IP addresses), thereby revealing the identity of the devices.

In Figure 15, we illustrate a few other deployment options that are suitable for real-world use cases. In the first scenario, an IoT user may leverage a trusted third party service such as a VPN server to implement iPET. The home gateway establishes a TLS connection with the remote VPN server, and iPET logic would be implemented at the home gateway (i.e., $\mathcal{R}_0$) and the VPN server as shown in Figure 15a. Second, we illustrate a scenario where anonymous onion-routing networks such as Tor [37] may be used. iPET can indeed be implemented as a Tor pluggable-transport [48] where the perturbed traffic is routed through a Tor *circuit* consisting of multiple peers as shown in Figure 15b. While a user may not have to trust one single entity, it comes at the cost of additional bandwidth overheads due to onion-routing encapsulation. Finally, an IoT user can consider a P2P (peer-to-peer) network of IoT gateway devices that support Trusted Execution Environments (TEEs), e.g., Intel SGX [26]. Like Tor circuits, an IoT gateway first creates a circuit of multiple peers, each of which implements iPET Logic as

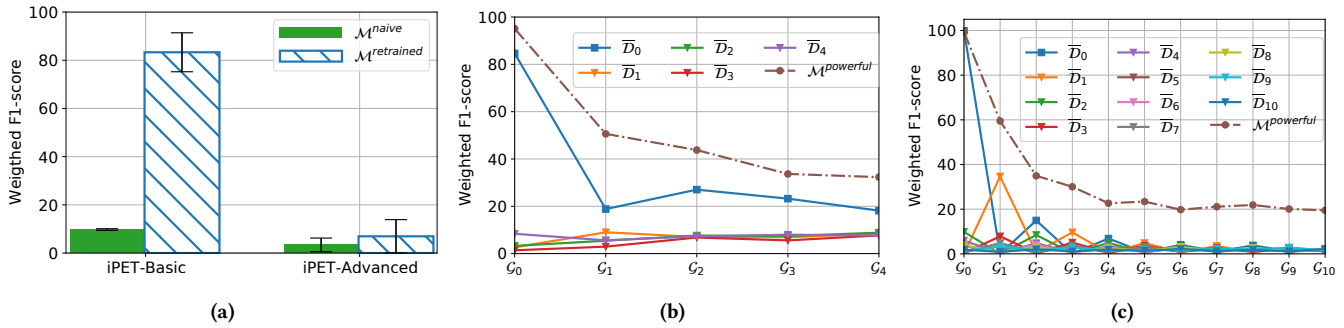
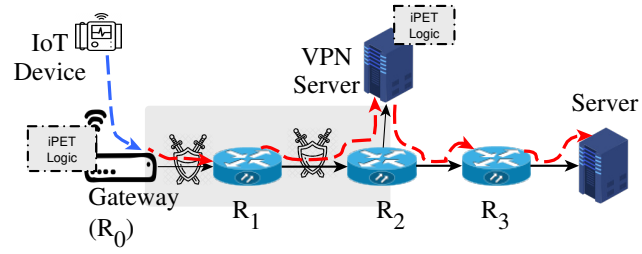
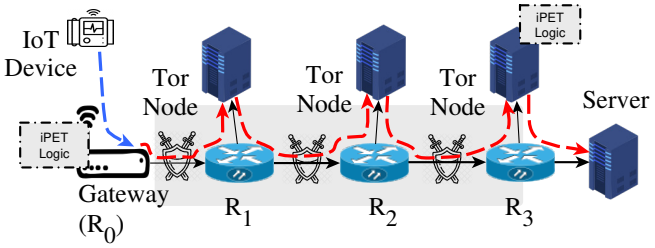


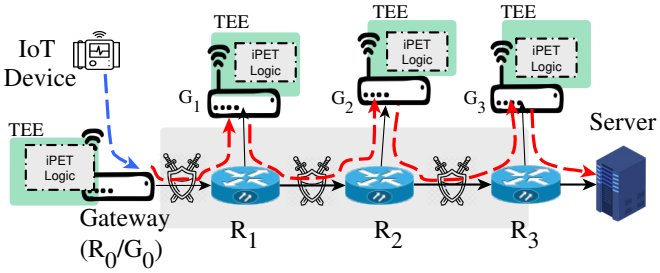
Figure 14: (a) iPET-Basic and iPET-Advanced against $\mathcal{M}^{\text{naive}}$ and $\mathcal{M}^{\text{retrained}}$. (b) Fingerprinting accuracy of attack discriminators $\overline{\mathcal{D}_j}$ and $\mathcal{M}^{\text{powerful}}$ (Powerful attacker), against different user generators $\mathcal{G}_i$ on Real-world dataset, and (c) Synthetic dataset.



(a) IoT user has access to a VPN server that implements iPET logic.



(b) IoT user has access to a modified Tor network that implements iPET logic as a pluggable transport layer.



(c) IoT user has access to a P2P network of IoT gateways that implement iPET logic.

Figure 15: iPET Deployment Scenarios.

shown in Figure 15c. Privacy sensitive operations such as encryption/decryption, traffic perturbations and routing can be performed inside the TEE. This option simplifies the design as it eliminates the need for complex protocols such as onion-encryption but instead relies on hardware-backed privacy guarantees provided by the TEE.

iPET on IoT Devices. The unique constraints of IoT devices (low power, compute, and storage capacity) have posed challenges in running ML models directly on the IoT hardware. However, recent developments in the TinyML space [9, 18, 41, 60] demonstrate the potential to perform ML inference on IoT devices. By design, iPET pre-trains compact device-specific traffic generators, which can further help with direct deployment on the devices. iPET using TinyML is an interesting future direction to explore.

iPET Against Website Fingerprinting. While iPET defends against *device fingerprinting* attacks, a relevant question is if iPET can be applied to other domains, for example, to defend against website fingerprinting attacks. The diverse functionalities of IoT devices lead to distinct device traffic characteristics, making device fingerprinting different from problems like website fingerprinting. For instance, a smart plug and a smart camera will generate distinct traffic rates compared to, say, web browsing, where most website visits may not differ substantially (in rate). Therefore, both fingerprinting attacks and defenses may work differently for IoT devices and websites. The applicability of iPET to other problems is worth studying separately in future.

7 RELATED WORKS

User traffic identification attacks are well-known, and several research works have proposed fingerprinting attacks that identify websites [4, 6, 16, 31, 34, 46, 50, 58], webpages [45, 57], videos [24, 29, 43, 62], browsers [23, 56], and even newer protocols [8] from encrypted communications. In the widely studied website fingerprinting attack, an attacker identifies the websites visited by a user over a network, with recent research works exploiting algorithms from machine learning and deep learning.

Several defenses [6, 20, 31, 35, 39, 46, 58, 59, 62] have been proposed to counter user traffic identification attacks, and website fingerprinting in particular, and all of them use some form of traffic

perturbation. Early countermeasures [6, 20, 35, 58, 59] for website fingerprinting used anonymous communication (e.g., Tor, VPN) in conjunction with a traffic perturbation algorithm. However, protecting against attackers [4, 31, 46] that use Machine Learning (ML) and Deep Learning (DL) has been a challenge. Recent countermeasures for website fingerprinting [32, 39] use DL-based traffic perturbations to deceive the ML/DL classifiers used by the attacker. These countermeasures add perturbations such that the perturbed traffic flows get incorrectly classified by the attacker's ML model. A key challenge is that an attacker can further retrain to learn the added perturbations. In this context, Mockingbird [39], a website fingerprinting defense, proposed the use of a semi-targeted objective function (i.e., chooses a set of target labels for misclassifying a datapoint) to increase the stochasticity of the perturbations, to counter retraining attackers.

In the context of fingerprinting IoT devices, many research works [3, 10, 13, 25, 27, 44, 47, 53, 55] have shown that it is possible to identify the devices with high accuracy, highlighting their vulnerability to passive network attacks. These attacks use ML classifiers trained on network traffic to identify the IoT devices, but assume that the IoT user has no defense in place. Trimananda et. al. [55] propose MTU padding as a potential defense against a similar attacker, however our evaluations in Section 5.1 show that a retraining adversary $\mathcal{M}_{\text{seq}}^{\text{retrained}}$ can easily defeat such a defense with high accuracy (92.4%). Our work is also related to the literature on device *activity* fingerprinting. These works [1, 3] have shown that IoT device activity (e.g., the on/off state of the devices) can be fingerprinted and propose traffic obfuscation techniques to prevent device activity fingerprinting. Although these works do not directly pertain to our work, the traffic obfuscation algorithms proposed could be applicable in preventing device fingerprinting attacks. Unlike iPET, which attempts to learn device patterns and generate device specific perturbations, their traffic obfuscation algorithm is oblivious to the device characteristics and generates constant rate perturbations. Moreover, the aforementioned solutions do not assume a strong attacker who retrains the perturbed traffic as a counter strategy. We push the boundary of defense in the IoT domain by proposing the first deep learning-based perturbation generator that is resilient to powerful attackers.

8 CONCLUSIONS

We developed and presented iPET, a new defense solution against IoT device fingerprinting attacks that is resilient to different attack capabilities. iPET uses a novel adversarial ML-based perturbation technique to effectively reduce the accuracy of fingerprinting attacks; our experiments demonstrate a reduction of fingerprinting accuracy from a very high 96.36% to an ineffective $\approx$ 17%, at a very low bandwidth overhead. Further, different from previous works, we considered a powerful attacker with knowledge of the iPET model's architecture and parameters; and our experiments show that such an attacker achieves only 50% accuracy against iPET. We also show that iPET generalizes well against different fingerprinting models (in existing studies) that an attacker may use.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their valuable feedback. This work was supported by MOE AcRF Tier 1 Grant T1 251RES2104; Huawei Research Grant TC20211206645; and Intra-CREATE Seed Collaboration Grant NRF2021-ITS007-0010. This research has also been supported by the National Research Foundation, Prime Minister's Office, Singapore under its Corporate Laboratory@University Scheme, National University of Singapore, and Singapore Telecommunications Ltd.

REFERENCES

[1] Noah J. Apthorpe, Danny Yuxing Huang, Dillon Reisman, Arvind Narayanan, and Nick Feamster. 2019. Keeping the Smart Home Private with Smart(er) IoT Traffic Shaping. *Proc. on Privacy Enhancing Technologies* (2019).
[2] Noah J. Apthorpe, Dillon Reisman, and Nick Feamster. 2017. A Smart Home is No Castle: Privacy Vulnerabilities of Encrypted IoT Traffic. *CoRR* (2017). http://arxiv.org/abs/1705.06805
[3] Noah J. Apthorpe, Dillon Reisman, Srikanth Sundaresan, Arvind Narayanan, and Nick Feamster. 2017. Spying on the Smart Home: Privacy Attacks and Defenses on Encrypted IoT Traffic. *CoRR* (2017). http://arxiv.org/abs/1708.05044
[4] Sanjit Bhat, David Lu, Albert Kwon, and Srinivas Devadas. 2019. Var-CNN: A Data-Efficient Website Fingerprinting Attack Based on Deep Learning. *Proc. on Privacy Enhancing Technologies* (2019).
[5] Leo Breiman. 2001. Random Forests. *Machine learning* 45, 1 (2001), 5–32.
[6] Xiang Cai, Xin Cheng Zhang, Brijesh Joshi, and Rob Johnson. 2012. Touching from a Distance: Website Fingerprinting Attacks and Defenses. In *ACM Conf. on Computer and Communications Security*.
[7] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS 2014 Workshop on Deep Learning, December 2014*.
[8] Levente Csikor, Himanshu Singh, Min Suk Kang, and Dinil Mon Divakaran. 2021. Privacy of DNS-over-HTTPS: Requiem for a Dream?. In *IEEE European Symposium on Security and Privacy*.
[9] Robert David, Jared Duke, Advait Jain, Vijay Janapa Reddi, Nat Jeffries, Jian Li, Nick Kreeger, Ian Nappier, Meghna Natraj, Tiezhen Wang, et al. 2021. Tensorflow lite micro: Embedded machine learning for tinyml systems. *Proceedings of Machine Learning and Systems* 3 (2021), 800–811.
[10] Antônio J. de Pinheiro, Jeandro M. Bezerra, Caio A. P. Burgardt, and Divanilson R. Campelo. 2019. Identifying IoT devices and events based on packet length from encrypted traffic. *Computer Communications* (2019). https://doi.org/10.1016/j.comcom.2019.05.012
[11] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. 2004. Tor: The Second-Generation Onion Router. In *USENIX Security Symposium*. http://www.usenix.org/publications/library/proceedings/sec04/tech/dingledine.html
[12] Dinil Mon Divakaran, Hema A. Murthy, and Timothy A. Gonsalves. 2006. Traffic Modeling and Classification Using Packet Train Length and Packet Train Size. In *Autonomic Principles of IP Operations and Management*.
[13] 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 *ACM Asia Conf. on Computer and Communications*. https://doi.org/10.1145/3320269.3384732
[14] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. In *International Conf. on Learning Representations (Poster)*.
[15] Google. 2022. *Google Transparency Report*. Available online at: https://transparencyreport.google.com/https/overview.
[16] Jamie Hayes and George Danezis. 2016. k-fingerprinting: A Robust Scalable Website Fingerprinting Technique. In *USENIX Security Symposium*.
[17] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.
[18] Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. 2021. Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks. *J. Mach. Learn. Res.* 22, 241 (2021), 1–124.
[19] Rob Jansen and Nicholas Hopper. 2012. Shadow: Running Tor in a Box for Accurate and Efficient Experimentation. In *Network and Distributed System Security Symposium*. https://www.ndss-symposium.org/ndss2012/shadow-running-tor-box-accurate-and-efficient-experimentation
[20] Marc Juárez, Mohsen Imani, Mike Perry, Claudia Díaz, and Matthew Wright. 2015. WTF-PAD: Toward an Efficient Website Fingerprinting Defense for Tor. *CoRR* (2015).
[21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2017. Imagenet classification with deep convolutional neural networks. *Commun. ACM* 60, 6 (2017), 84–90.

[22] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. 2016. Adversarial Machine Learning at scale. *arXiv preprint arXiv:1611.01236* (2016).

[23] Pierre Laperdrix, Walter Rudametkin, and Benoit Baudry. 2015. Mitigating Browser Fingerprint Tracking: Multi-level Reconfiguration and Diversification. In *IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. https://doi.org/10.1109/SEAMS.2015.18

[24] Feng Li, Jae Won Chung, and Mark Claypool. 2018. *Silhouette: Identifying YouTube Video Flows from Encrypted Traffic*.

[25] Manuel López Martín, Belén Carro, Antonio Sánchez-Esguevillas, and Jaime Lloret. 2017. Network Traffic Classifier With Convolutional and Recurrent Neural Networks for Internet of Things. *IEEE Access* (2017). https://doi.org/10.1109/ACCESS.2017.2747560

[26] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V. Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R. Savagaonkar. 2013. Innovative instructions and software model for isolated execution. In *Workshop on Hardware and Architectural Support*. https://doi.org/10.1145/2487726.2488368

[27] Yair Meidan, Michael Bohadana, Asaf Shabtai, Martín Ochoa, Nils Ole Tippenhauer, Juan David Guarnizo, and Yuval Elovici. 2017. Detection of Unauthorized IoT Devices Using Machine Learning Techniques. *CoRR* (2017).

[28] Francesca Meneghello, Matteo Calore, Daniel Zucchetto, Michele Polese, and Andrea Zanella. 2019. IoT: Internet of Threats? A Survey of Practical Security Vulnerabilities in Real IoT Devices. *IEEE Internet of Things Journal* (2019). https://doi.org/10.1109/JIOT.2019.2935189

[29] Gargi Mitra, Prasanna Karthik Vairam, Patanjali Slpsk, and Nitin Chandrachoodan. 2019. White Mirror: Leaking Sensitive Information from Interactive Netflix Movies using Encrypted Traffic Analysis. In *ACM SIGCOMM Conf. Posters and Demos*.

[30] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. 2017. Universal Adversarial Perturbations. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*.

[31] Milad Nasr, Alireza Bahramali, and Amir Houmansadr. 2018. DeepCorr: Strong Flow Correlation Attacks on Tor Using Deep Learning. In *ACM SIGSAC Conf. on Computer and Communications Security*.

[32] Milad Nasr, Alireza Bahramali, and Amir Houmansadr. 2021. Defeating DNN-Based Traffic Analysis Systems in Real-Time With Blind Adversarial Perturbations. In *USENIX Security Symposium*. https://www.usenix.org/conference/usenixsecurity21/presentation/nasr

[33] Peter Newman. 2020. *The Internet of Things 2020*. Available online at: https://www.businessinsider.com/internet-of-things-report.

[34] Andriy Panchenko, Lukas Niessen, Andreas Zinnen, and Thomas Engel. 2011. Website fingerprinting in onion routing based anonymization networks. In *ACM Workshop on Privacy in the Electronic Society*.

[35] Mike Perry. 2013. *Experimental Defense for Website Traffic Fingerprinting*. Available online at: https://blog.torproject.org/critique-website-traffic-fingerprinting-attacks.

[36] Omid Poursaeed, Isay Katsman, Bicheng Gao, and Serge J. Belongie. 2018. Generative Adversarial Perturbations. In *IEEE Conf. on Computer Vision and Pattern Recognition*. http://openaccess.thecvf.com/content_cvpr_2018/html/Poursaeed_Generative_Adversarial_Perturbations_CVPR_2018_paper.html

[37] Tor Project. 2021. *tor - Tor's source code*. Available online at: https://gitweb.torproject.org/tor.git/tree/?h=tor-0.4.4.8.

[38] Tor Project. 2022. *Tor Padding Specification*. Available online at: https://github.com/torproject/torspec/blob/79da008392caed38736c73d839df7aa80628b645/padding-spec.txt.

[39] Mohammad Saidur Rahman, Mohsen Imani, Nate Mathews, and Matthew Wright. 2021. Mockingbird: Defending Against Deep-Learning-Based Website Fingerprinting Attacks With Adversarial Traces. *IEEE Trans. on Information Forensics and Security* (2021).

[40] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1985. *Learning internal representations by error propagation*. Technical Report. California Univ San Diego La Jolla Inst for Cognitive Science.

[41] Manuele Rusci, Alessandro Capotondi, and Luca Benini. 2020. Memory-driven mixed low precision quantization for enabling deep network inference on microcontrollers. *Proceedings of Machine Learning and Systems* 2 (2020), 326–335.

[42] Mike Schuster and Kuldip K Paliwal. 1997. Bidirectional Recurrent Neural Networks. *IEEE transactions on Signal Processing* 45, 11 (1997), 2673–2681.

[43] Roei Schuster, Vitaly Shmatikov, and Eran Tromer. 2017. Beauty and the Burst: Remote Identification of Encrypted Video Streams. In *USENIX Security Symposium*.

[44] Mustafizur R. Shahid, Gregory Blanc, Zonghua Zhang, and Hervé Debar. 2018. IoT Devices Recognition Through Network Traffic Analysis. In *IEEE International Conf. on Big Data*. https://doi.org/10.1109/BigData.2018.8622243

[45] Meng Shen, Yiting Liu, Liehuang Zhu, Xiaojiang Du, and Jiankun Hu. 2020. Fine-Grained Webpage Fingerprinting Using Only Packet Length Information of Encrypted Traffic. *IEEE Trans. on Information Forensics and Security* (2020).

[46] Payap Sirinam, Mohsen Imani, Marc Juárez, and Matthew Wright. 2018. Deep Fingerprinting: Undermining Website Fingerprinting Defenses with Deep Learning. In *ACM SIGSAC Conf. on Computer and Communications Security*.

[47] Arunan Sivanathan, Hassan Habibi Gharakheili, Franco Loi, Adam Radford, Chamith Wijenayake, Arun Vishwanath, and Vijay Sivaraman. 2019. Classifying IoT Devices in Smart Environments Using Network Traffic Characteristics. *IEEE Trans. on Mobile Computing* (2019). https://doi.org/10.1109/TMC.2018.2866249

[48] Shari Steele. 2016. *Tor at the Heart: Bridges and Pluggable Transports* . Available online at: https://blog.torproject.org/tor-heart-bridges-and-pluggable-transports/.

[49] Kalupahana Liyanage Kushan Sudheera, Dinil Mon Divakaran, Rhishi Pratap Singh, and Mohan Gurusamy. 2021. ADEPT: Detection and identification of correlated attack stages in IoT networks. *IEEE Internet of Things Journal* (2021), 6591–6607. https://doi.org/10.1109/JIOT.2021.3055937

[50] Qixiang Sun, Daniel R Simon, Yi-Min Wang, Wilf Russell, Venkata N Padmanabhan, and Lili Qiu. 2002. Statistical identification of encrypted web browsing traffic. In *IEEE Symposium on Security and Privacy*.

[51] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2014. Intriguing Properties of Neural Networks. *CoRR* (2014).

[52] TensorFlow. 2020. *TensorFlow v2.5.0*. Available online at: https://www.tensorflow.org/versions/r2.5/api_docs/python/tf.

[53] Vijayanand Thangavelu, Dinil Mon Divakaran, Rishi Sairam, Suman Sankar Bhunia, and Mohan Gurusamy. 2019. DEFT: A Distributed IoT Fingerprinting Technique. *IEEE Internet Things Journal* (2019). https://doi.org/10.1109/JIOT.2018.2865604

[54] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. 2017. Ensemble adversarial training: Attacks and defenses. *arXiv preprint arXiv:1705.07204* (2017).

[55] Rahmadi Trimananda, Janus Varmarken, Athina Markopoulou, and Brian Demsky. 2020. Packet-Level Signatures for Smart Home Devices. In *Network and Distributed System Security Symposium*. https://www.ndss-symposium.org/ndss-paper/packet-level-signatures-for-smart-home-devices/

[56] Antoine Vastel, Pierre Laperdrix, Walter Rudametkin, and Romain Rouvoy. 2018. FP-STALKER: Tracking Browser Fingerprint Evolutions. In *IEEE Symposium on Security and Privacy*.

[57] Kailong Wang, Junzhe Zhang, Guangdong Bai, Ryan Ko, and Jin Song Dong. 2021. It's Not Just the Site, It's the Contents: Intra-domain Fingerprinting Social Media Websites Through CDN Bursts. In *The Web Conference*.

[58] Tao Wang, Xiang Cai, Rishab Nithyanand, Rob Johnson, and Ian Goldberg. 2014. Effective Attacks and Provable Defenses for Website Fingerprinting. In *USENIX Security Symposium*.

[59] Tao Wang and Ian Goldberg. 2017. Walkie-Talkie: An Efficient Defense Against Passive Website Fingerprinting Attacks. In *USENIX Security Symposium*.

[60] Xiaying Wang, Michele Magno, Lukas Cavigelli, and Luca Benini. 2020. FANN-on-MCU: An open-source toolkit for energy-efficient neural network inference at the edge of the Internet of Things. *IEEE Internet of Things Journal* 7, 5 (2020), 4403–4417.

[61] Yu Wang, Yang Xiang, Jun Zhang, Wanlei Zhou, Guiyi Wei, and Laurence T. Yang. 2014. Internet Traffic Classification Using Constrained Clustering. *IEEE Transactions on Parallel and Distributed Systems* 25, 11 (2014), 2932–2943. https://doi.org/10.1109/TPDS.2013.307

[62] Xiaokuan Zhang, Jihun Hamm, Michael K. Reiter, and Yinqian Zhang. 2019. Statistical Privacy for Streaming Traffic. In *Network and Distributed System Security Symposium*.

A AGGREGATE FEATURES MODEL

We list the features that $\mathcal{M}_{agg}$ adversary extracts in Table 4. We use "statistics" to cumulatively denote the minimum, maximum and mean values of the feature and "#" to denote the number of occurrences of that feature. As explained in Section 2.2, the features are grouped by different protocols (e.g., TCP, UDP, NTP, etc.) and are aggregated over a session of some configured length.

An attacker in the undefended scenario has access to all the features listed in Table 4. However, in some deployment scenarios (Section 6), the packet protocol may not be known. For instance, a VPN might overlay all UDP and TCP packets over a single TCP connection. In that case, several protocol-specific features are unavailable, and instead, the attacker simply computes the total count of packets, packet length statistics, and the flow duration. The flow duration is the difference between the time at which the last and the first packet were observed in a session. In order to maximise the information drawn from these features, we group them based on the direction of the packets.

Protocol	Feature Extracted
DNS	# DNS queries # DNS errors in reply code # DNS 'IN' class queries Average DNS response time # Packets Packet Length statistics
HTTP	# TCP keep Alive # Unique HTTP status codes # Unique HTTP request methods # Unique HTTP request URI # Packets Packet Length statistics
All other protocols	# Packets Packet Length statistics

Table 4: List of all features utilized. Other protocols include: TCP, UDP, ICMP, TLS, SSDP, NTP, STUN, GQUIC.

B $\mathcal{M}_{\text{agg}}$ VERSUS $\mathcal{M}_{\text{seq}}$

Anonymous communication tools such as Tor direct traffic through a peer-to-peer overlay network to obscure many of the packet headers that the $\mathcal{M}_{\text{agg}}$ classifier uses. Table 5 shows the weighted F1-score achieved by $\mathcal{M}_{\text{agg}}$ and $\mathcal{M}_{\text{seq}}$ against Tor and undefended traffic belonging to 30 IoT devices (Section 5 describes the dataset). Note the drop in accuracy of $\mathcal{M}_{\text{agg}}$ compared to $\mathcal{M}_{\text{seq}}$.

	$\mathcal{M}_{\text{agg}}$	$\mathcal{M}_{\text{seq:CNN}}$
Undefended	100	96.36
Tor	65.3	87.1
iPET	**15.8**	**17.74**

Table 5: Weighted F1-score achieved by attacker on IoT devices with at least 0.5% support.