Virginia Commonwealth University
VCU Scholars Compass

Theses and Dissertations Graduate School

2022

Machine Learning (ML) - assisted tools for enhancing security and privacy of edge devices

Santosh Kumar Nukavarapu
Virginia Commonwealth University

Follow this and additional works at: https://scholarscompass.vcu.edu/etd

Part of the Computer Engineering Commons

© The Author

Downloaded from
https://scholarscompass.vcu.edu/etd/6958

This Dissertation is brought to you for free and open access by the Graduate School at VCU Scholars Compass. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of VCU Scholars Compass. For more information, please contact libcompass@vcu.edu.

©Santosh, March 2022

All Rights Reserved.

MACHINE LEARNING (ML) - ASSISTED TOOLS FOR ENHANCING SECURITY AND PRIVACY OF EDGE DEVICES

A Thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at Virginia Commonwealth University.

by

SANTOSH KUMAR NUKAVARAPU

Director: Dr. Tamer Nadeem,

Associate Professor, Department of Computer Science

Virginia Commonwealth University

Richmond, Virginia

March, 2022

Acknowledgements

I express my sincere gratitude to my advisor, Prof. Tameer Nadeem, for his continuous inspiration and guidance in my research work. Prof. Tameer Nadeem has provided key directional ideas in this research work. His consistent guidance, expertise, and dedication to providing rigorous review comments helped me improve my subject knowledge and technical writing skills. I am very grateful to him for providing me the opportunity to explore and research such an interesting and imperative subject. I would like to express my sincere thanks to my Ph.D. Committee members, Prof.Ravi Mukkamala, Prof.Wei Cheng, Prof.Irfan Ahmed, Prof.Preetam Ghosh, and Prof.Tamer Nadeem for their insightful comments and suggestions to ensure the right path towards my research progress. I am also thankful to the Professors, Staff, and Students of the Department of Computer Science, VCU College of Engineering, for their support and help. All of them contributed to an excellent and inspiring working atmosphere. I am also grateful and thankful to Mr.Mohammad Ayyat and Dr.Mostafa Abdulla Uddin for their collaboration on DDAS/iBranchy and PrivacyGuard projects. Special thanks go to my colleagues Ms.Hanna B Passandi, Mr.Mohammad Ayyat, and Ms.Norah Alobaidan for their support and encouragement during this research work. I owe big thanks to my family members for their love, endless patience, and encouragement. Finally, I would like to thank God and my family for their blessings, and constant support.

TABLE OF CONTENTS

List of Algorithms

LIST OF FIGURES

Abstract

MACHINE LEARNING (ML) - ASSISTED TOOLS FOR ENHANCING SECURITY AND PRIVACY OF EDGE DEVICES

By Santosh Kumar Nukavarapu

A Thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at Virginia Commonwealth University.

Virginia Commonwealth University, 2022.

Director: Dr. Tamer Nadeem,
Associate Professor, Department of Computer Science

The rapid growth of edge-based IoT devices, their use cases, and autonomous communication has created new challenges with privacy and security. Side-channel attacks are one of the examples of security and privacy vulnerabilities that can cause inference at Internet-Service Provider (ISP) and local Wi-Fi networks. Such an attack would leak user's sensitive information such as home occupancy, medical activity, and daily routines. Another example is that these devices have weak authentication and low encryption standards, making them an easy target for malware-based attacks such as denial of service or launching other network attacks using these infected devices. This thesis dissertation explored ML-assisted tools to secure the devices from network-based security and privacy inference attacks. In this thesis, one of our components, securing edge devices against malware-based attacks or anomalies, focused on building an ML-assisted security service and evaluating it for different real-world conditions such as robustness to noise, adaptability to new devices, and less resource-intensive deployment. Moreover, given the recent trend of adversarial attacks against machine learning models, we designed a novel edge-based adversarial attack against edge-based machine learning models and explored techniques to make edge-based network security services robust. Another component of this thesis focused on building a privacy-preserving service to obfuscate and minimize the privacy inference on network communication. The main objective of this component was to evaluate different privacy-preserving techniques such as traffic shaping and injecting synthetic network traffic for countering the effectiveness of side-channel attacks.

CHAPTER 1

INTRODUCTION

1.1 Background

With the phenomenal growth of IoT devices at the edge, many exciting applications are being enabled, such as remote health care [1], augmented reality [2] and video analytics [3]. The majority of these devices are connected to the remote servers and communicate periodically or continuously even without any user intervention [4]. This cloud-connected architecture has enabled remote monitoring, flexible user control, and automatic updates of IoT devices. Moreover, these devices can be connected to third platforms through their remote servers for supporting smart home automation scenarios using third-party platforms such as IFTTT[5]. The rapid growth of edge-based IoT devices, their use cases, and autonomous communication has created new challenges with privacy and security. Side-channel attacks are one of the examples of security and privacy attacks that can cause inference at Internet-Service Provider (ISP) and local Wi-Fi networks. Such an attack would leak users' sensitive information such as home occupancy, medical activity, and daily routines. Another example is that these devices have weak authentication and low encryption standards, making them an easy target for malware-based attacks such as denial of service or launching other network attacks using these infected devices.

1.2 Motivation

In recent years, machine learning has been adopted in networking and security domains for classification and regression tasks given its abilities such as high performance with accuracy, outlier detections, ability to recognize hidden patterns in data, and less human intervention. To secure IoT devices from different types of network attacks [6, 7], the IoT/edge applications typically exploit machine learning models such as Deep Neural Network (DNN) models for different security functionalities such as traffic classification, device identification, and anomaly detection. However, there are multiple challenges in building machine learning tools for edge-based devices, such as the availability of less labeled data due to privacy issues, the ability to incrementally update existing models for changes in device data such as concept drift, domain expertise for feature engineering. Moreover, the performance of these models when running on edge-based IoT devices will be significantly impacted by the limitations of the device resources which will reflect on the performance of these devices. Therefore, it is highly desirable to develop techniques to optimally accelerate the inference computations of DNN models in order to enable real-time applications and conserve energy for edge devices/IoT.

1.3 Research Objectives

In this thesis, we explored the design and implementation of an ML-assisted security and privacy framework to counter IoT-based network attacks. To realize this, we build ML-assisted Security and Privacy services with multiple sub-components for securing the edge devices against security and privacy attacks. In the first part of the proposal, we explore developing a security service tailored for real-world conditions edge-based IoT conditions such as robustness to noise, adaptability to new devices, and less resource-intensive deployment. Moreover, given the recent trend of adversarial attacks against machine learning models where an adversary can craft malicious inputs such as malware to be classified as benign, we want to explore how to make security services robust. The other significant part of this proposal is to design a privacy-preserving service to obfuscate and minimize the privacy inference on network communication. The main objective of this component is to evaluate different privacy-preserving techniques such as traffic shaping and injecting synthetic network traffic for countering the effectiveness of side-channel attacks. Given, many of these devices, such as baby monitoring cameras, provide sensitive operations and have strict QoS requirements. Therefore, the edge-based IoT infrastructure should support different security, privacy, and QoS requirements by enabling network-wide services. Furthermore, given the dynamic network characteristics of these devices, such as mobility, resource-constrained needs (sleep periods), the network services must be flexible. We explore different techniques to make our privacy service's flexible and adaptable for different dynamic environment conditions such as network latency and location using a programmable approach.

Most importantly, in this thesis, we focus on the ability of machine learning-based models to assist in building these services to enhance the privacy and security of edge-based IoT devices. One of the ML models that we explore in this thesis is Generative adversarial Networks (GANs) that have shown excellent ability to mimic the training data distribution and automatically reverse engineer the training data structure without any feature engineering. GANs are trained using an adversarial approach where two networks compete against each other through a adversarial training; this process has been shown to build better generative and discriminative models. We believe that this feature of GANs can help to solve multiple challenges in the IoT domain. For example, GANs could create new synthetic data that can help with the privacy challenges, and the discriminative features will help build better anomaly detection-based systems. Based on our objectives which we explain in later chapters, we also explore other ML models based on Knowledge distillation, Dynamic Deep Neural Networks and Continual learning to build lightweight, flexible, and adaptable machine learning assisted services required for edge-based scenarios.

1.4 Outline of Thesis

We organize the rest of the document as follows: Chapter 2 discusses the different network-based attacks on IoT devices , describes the attack threat model addressed in our work and machine learning model details. We then discuss the vision of our framework consisting of network-based security and privacy service services with multiple sub-components in Chapter 3. Chapter 4 focuses on building an ML-assisted security service and evaluating it for different real-world edge-based- IoT scenarios. Chapter 5 and Chapter 6 discuss our design for lightweight ML-assisted security service for edge-based scenarios using Dynamic Deep Neural Networks and their related vulnerabilities and defenses. Chapter 7 discusses our other component that focuses on building a privacy-preserving service to obfuscate and minimize the privacy inference on network communication. Chapter 8 evaluates GANs for network packet Generation, our first step towards creating a GAN-based synthetic Network Traffic Generation framework that can assist with the privacy-preserving service. Finally, we summarize our contributions and discuss our future work plan.

CHAPTER 2

BACKGROUND

2.1 IoT Network based Attacks

In recent years, edge-based IoT devices are found vulnerable for different types of network-based attacks [6]. One of the main reasons for this vulnerability is the design and implementation of the network model of these devices. IoT devices are generally known to have two network subsystems, a keep-alive subsystem and activity-driven subsystem [8, 9]. The keep-alive subsystem keeps the device connected to the cloud server and enables the device to receive remote communications [8] while the activity-driven subsystem sends the periodic application data such as sensor or user activity events. This type of autonomous network architecture significantly differs from the non-IoT devices such as laptops and smartphones that generally communicate to remote servers only on specific user-based application activities. A recent study showed that IoT devices communicate to cloud servers over 22 days without any user interactions [4].

This type of autonomous communication of the network model makes IoT devices ubiquitous, thus supporting many beneficial use cases such as home monitoring and home automation with minimal or no user intervention. However, at the same time, this type of autonomous communication has made these devices vulnerable to different side-channel and availability-based attacks. Moreover, given that these network models have weak authentication and cipher suites during the network communication, the IoT devices have become vulnerable to many other attacks such as replay, malware, etc. In this chapter, we first discuss the different types of IoT-based network attacks, briefly explain the attacks that we address in our work, and finally discuss our framework design.

There are different types of network attacks found on IoT devices based on a recent work of authors who categorized the taxonomy of IoT-based network attacks [6]. The IoT-based network attacks are classified as passive attacks or active attacks. In passive attacks, the attacker either eavesdrops on the communication between the device and a user or server to infer device-specific information to engineer device-specific attacks or analyzes the network traffic for fingerprinting devices and activities. For active attacks, the attacker performs various network-based activities such as probing, scanning, or other activities involving network communication with the devices to launch attacks. For example, the attacker can try to get metadata about the device by probing, scanning, etc., to find device-specific vulnerabilities. Similarly, the attacker can send old packets to devices to perform replay-based attacks, given these devices have weak authentication in their network stack implementation. Moreover, some of these attacks focus on denial of service either by dropping, setting early expiry to active certificates through fake NTP packets, or adding additional processing by increasing sleep periods by sending more packets. Finally, malware-based attacks infect IoT devices to launch further attacks such as distributed denial of service, and spoofing-based attacks extract sensitive information through packets.

2.2 The significance of IoT classification

The online network model of IoT devices makes them vulnerable to many network attacks. For example, an attacker can use a publicly available tool such as Shodan [10] to identify the different vulnerable IoT devices such as cameras connected to the internet and create bots for their distributed denial of service attacks (DDoS). Recently, Mirai, an IoT malware, had a network of massive 600,000 devices (bots) at its peak stage and infected more than 64,500 devices such as IP cameras, TV receivers, and printers within the first 20 hours of its emergence[11]. Moreover, this online network architecture can cause side-channel attacks for activity inference at ISP and local WiFi network [7]. The traffic traces of different activities of a device can have different signatures based on [7]. Also, the traffic signatures of the same device with different activities are different. Recently, researchers have shown different attack models for activity inference [7, 12].

Given that IoT devices have low resource constraints, many security solutions such as anti-malware solutions deployed on non-IoT devices such as traditional personal computers, laptops, and phones are ineffective for IoT devices. Moreover, unlike non-IoT devices, IoT devices are integrated with different third-party apps for automation purposes and have a larger attack surface due to more autonomous network connections. Given these issues,special network policies for securing IoT devices needs to be enforced and therefore having a classifier that can identify the device as IoT or Non-IoT is essential. The network community is looking to build security and privacy solutions at edge infrastructure to address the above issues. For example, authors in [13] deploy software-defined networking models for security purposes by isolating the network traffic through the CSS vulnerability database and IoT-based classifier for detecting anomalies. However, to implement and automate any such SDN-based solutions, it is essential to have an effective classification system that is not dependent on labeled data and manual feature extraction.

Some of the recent works have proposed convolutional neural networks (CNN) [14] for network traffic classification [15]. CNNs are a deep learning model that has have been generally used for image classification [16] and object detection [17]. A CNN based deep learning architecture uses convolution operation to extract features for spatial data such as images. Recently, researchers have shown the network traffic has spatial relationships similar to the images, which can be efficiently processed using CNNs [15]. However, like any other supervised deep learning model, CNN is impacted by labeled data availability. Recently CNNs as a classifier trained through a semi-supervised method based on Generative Adversarial Networks (GANs) [18] is emerging as a good model with less labeled data but high accuracy. As discussed in later chapters, we use

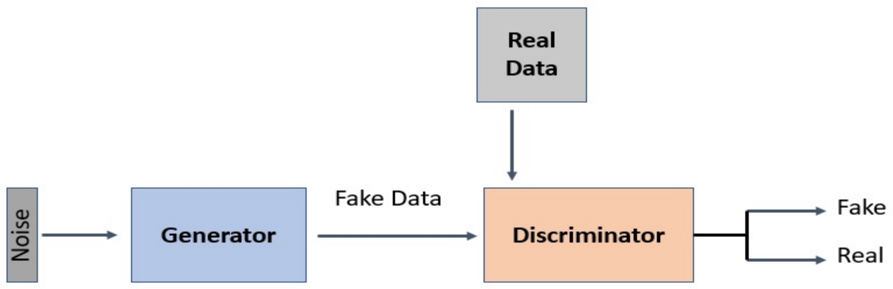


Figure 1: Generative Adversarial Network

Generative Adversarial Networks to build our components for edge devices' security and privacy.

2.3 Generative Adversarial Networks

For many complex generative tasks such as image generation, estimating the probability distribution of the features of the images is difficult. Generative adversarial networks (GANs) [18] are known to model unknown complex distributions such as image generation tasks. The GAN model consists of two neural networks, discriminator and Generator, that form a two-player min-max game where the Generator tries to generate fake samples. The discriminator tries to identify if the samples are from the training data (real) or generated from the Generator (fake). As shown in Figure 1, the generator model accepts as an input a random noise and then transforms this noise to a fake sample as an output. The min-max game forces the Generator to approximate the true distribution. Moreover, the training process reaches an optimum solution when the min-max game reaches a Nash Equilibrium when the Generator's fake samples are predicted as fake by the discriminator with a 50% probability. The GAN training consists of simultaneously training the Generator and the discriminator models on small batches of data known as mini-batch with a fixed size. Each iteration will train the GAN model with this fixed mini-batch of data points. For a training iteration, the

Generator is given a batch of random points from the latent space to output a batch of fake samples. This batch of fake samples is given to the discriminator to predict if they are real or fake. The discriminator's loss for predicting fake samples is given to the Generator, which updates its weights accordingly to generate better realistic samples. The generator training is called unsupervised training as the generator model is never exposed to the real data during training. After training, the Generator can be detached from the GAN model and can be independently used for generative tasks. Similarly, the discriminator model can be used for many image classification tasks [19, 20]. Very recently, image-based anomaly detection using GAN discriminator has been proposed [21].

CHAPTER 3

A SECURITY AND PRIVACY BASED FRAMEWORK FOR THE EDGE-BASED IOT DEVICES

3.1 Network attacks addressed in our work

This work focuses on addressing two types of attacks: passive traffic analysis-based side-channel attacks that can cause serious privacy inferences and the need for anomaly and malware detection to assist with malware-based distributed denial of service attacks. Below we describe the threat model of each of the network attacks briefly.

Privacy inference based side-channel attacks :

The broadcasting nature of WiFi exposes IoT and smart devices traffic to eavesdropping by adversaries and, consequently, various attacks. Several previous studies already showed that, even with WiFi encryption (e.g., WPA2), statistical analysis of side-channel information of WiFi traffic such as packet sizes, data rate, ratio of incoming to outgoing packets, inter-packet time, etc. could infer several user-related information such as user identity [22], user's online activities [23], and identification of applications used by the user [24]. The traffic analysis of major commercial IoT devices like Nest Camera, Amazon Echo (Personal Voice Assistant), Belkin Smart Plug is found vulnerable to activity inference such as user presence, device interaction and appliance usage[25]. As an example, Figure 2 a) captures the WiFi traffic of four mobile applications corresponding to four IoT devices, while 2 b) zooms on the traffic of the same-type devices. As shown, while the traffic patterns of the two Flux bulbs look very similar, different IoT devices have different traffic patterns that could be uniquely distinguished and easily correlated to its corresponding IoT device and even to a specific

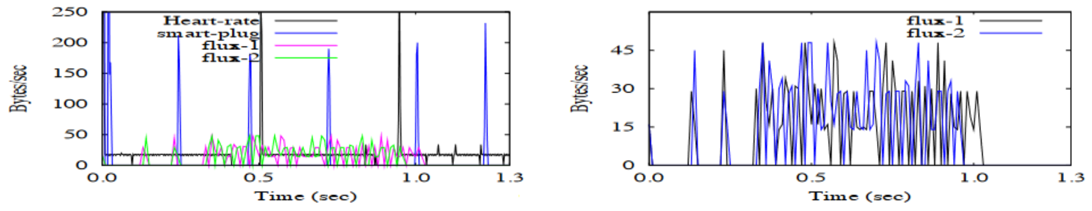


Figure 2: a) Traffic patterns of four different IoT devices: Pulse-Heart rate monitoring device, Elegato-plug device, and two Flux-lightbulb devices operating at different times. b) Zooming into the traffic of two Flux-lightbulb devices shows high similarity.

activity/status of the IoT device.

Malware and Anomaly detection :

The online network of IoT model devices makes them vulnerable to malware-based attacks. For example, an attacker can use a publicly available tool such as Shodan [10] to identify the different vulnerable IoT devices such as cameras connected to the internet and create bots for their distributed denial of service attacks (DDoS). Recently, Mirai, an IoT malware, had a network of massive 600,000 devices (bots) at its peak stage and infected more than 64,500 devices such as IP cameras, TV receivers, and printers within the first 20 hours of its emergence[11].

Specifically, our focus is on detecting the attack signatures of various IoT-specific malware and any new zero-day-based attacks whose signature is unknown. Therefore, the main objective in addressing malware attacks is to differentiate between benign devices and malicious malware-based network traffic. In the below section, we discuss our framework for countering both the side channel and malware-based attacks.

3.2 Framework Vision

Our vision is to design and implement ML-assisted security and privacy-based network services for edge-based IoT devices. We define a network service as a set of programmable APIs that dynamically define and configure different network operations (schemes) with configurable parameters. For example, a privacy-based service can have an API method to dynamically enable a privacy-preserving scheme to perform obfuscation operations such as delay, padding, or spoof network packets. In addition to this, our service API methods are flexible as they can be mapped to specific device type policies where a policy is a device type identification for a particular context (e.g., user location, time, battery level, network load, etc.). Similarly, a security-based service can have policies to isolate an IoT device if its network traffic is found infected by malware. To realize our vision, we first design network-based security and privacy services using ML assisted techniques and then implement them by extending and deploying SDN components (e.g., Open vSwitch [26]) on WiFi Access Points (in-home or campus environments) or Proxy server (in open and public WiFi Hotspots).

Figure 3 shows the overall design of our security and privacy framework. The security component is realized by building *iKnight* to detect different device types, malware, and anomalies. Next, the privacy part is realized by building multiple sub-components, that includes *PrivacyGuard* - a traffic shaping-based privacy-preserving service; *MirageNet* - a synthetic IoT network traffic generation tool and context-based privacy-preserving service that applies the appropriate privacy-preserving scheme based on context such as network, sensitivity and location. Next, we briefly discuss each of these components and their interdependence.

3.2.1 *iKnight* - A edge based security service for IoT devices

The main objective of the edge-based security service is to identify IoT devices, their device types (IoT vs. Non-IoT), malware, or anomalies. There are many challenges in building such a service due to the edge-based IoT device characteristics, less availability of labeled data, manual feature engineering for different types of devices from various vendors, ability to infer new devices without needing to re-train the model from scratch, ability to deploy on an edge-based device with fast inference speed and light resource consumption. Given these requirements, we build *iKnight* a multistage

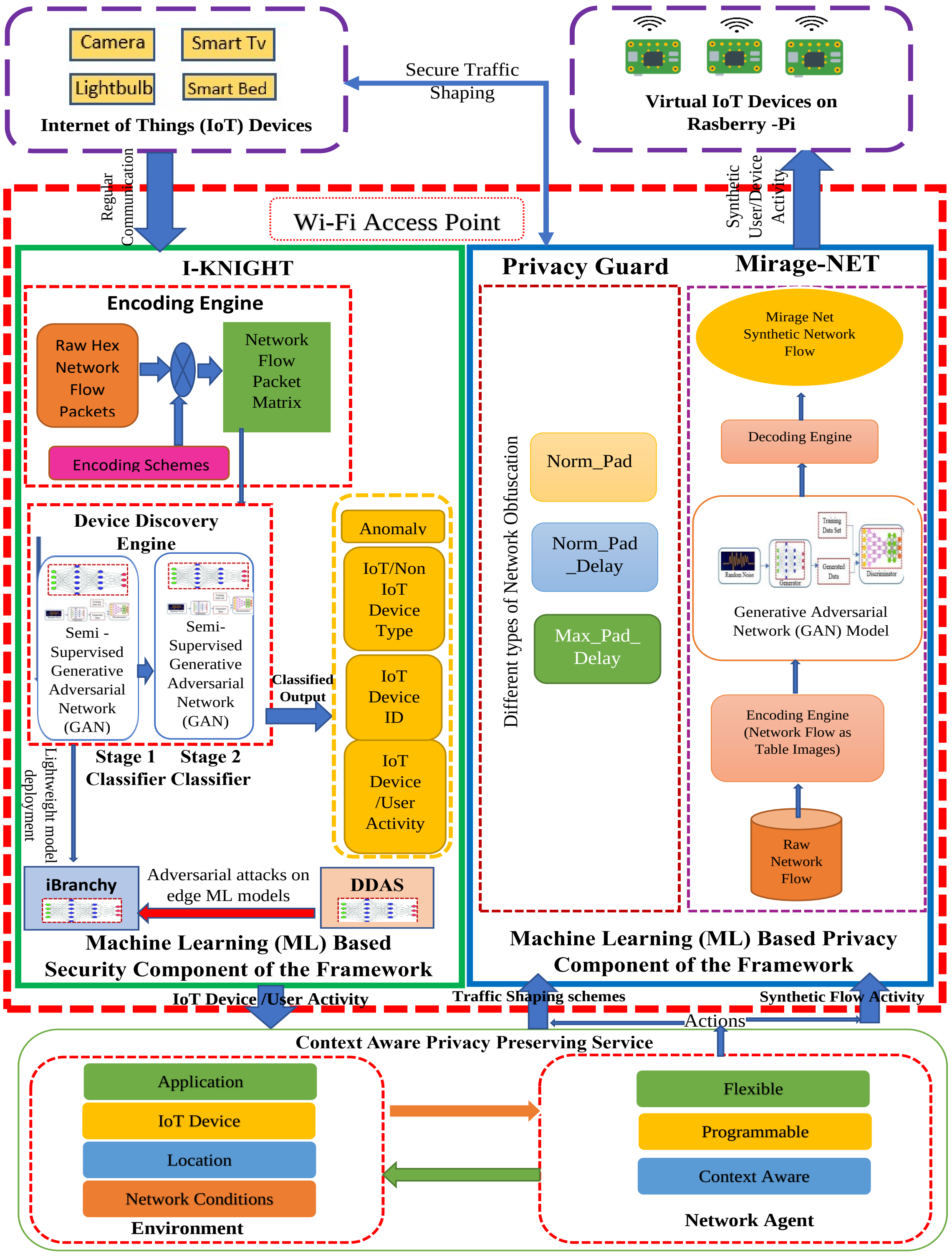

Camera
Smart Tv
Lightbulb
Smart Bed
Internet of Things (IoT) Devices
Secure Traffic Shaping
Virtual IoT Devices on Rasberry -Pi
Regular Communication
Synthetic User/Device Activity
Wi-Fi Access Point
I-KNIGHT
Encoding Engine
Raw Hex Network Flow Packets
Network Flow Packet Matrix
Encoding Schemes
Device Discovery Engine
Semi - Supervised Generative Adversarial Network (GAN)
Semi- Supervised Generative Adversarial Network (GAN)
Stage 1 Classifier
Stage 2 Classifier
Classified Output
Anomaly
IoT/Non IoT Device Type
IoT Device ID
IoT Device /User Activity
Lightweight model deployment
iBranchy
Adversarial attacks on edge ML models
DDAS
Machine Learning (ML) Based Security Component of the Framework
Privacy Guard
Mirage-NET
Different types of Network Obfuscation
Norm_Pad
Norm_Pad _Delay
Max_Pad_ Delay
Mirage Net Synthetic Network Flow
Decoding Engine
Generative Adversarial Network (GAN) Model
Encoding Engine (Network Flow as Table Images)
Raw Network Flow
Machine Learning (ML) Based Privacy Component of the Framework
IoT Device /User Activity
Traffic Shaping schemes
Synthetic Flow Activity
Actions
Context Aware Privacy Preserving Service
Application
IoT Device
Location
Network Conditions
Environment
Flexible
Programmable
Context Aware
Network Agent

Figure 3: Framework Vision

multi-class classifier-based service to guard the IoT infrastructure at the edge. We choose to develop the service using a Semi-Supervised GAN and other supporting ML techniques to implement the requirements for edge-based IoT classifiers. As shown in the framework diagram, the input to *iKnight* are the flows of the edge devices, and the output is the identified device/type/malware or an anomaly. We also built and evaluated a new approach to deploying lightweight model *iBranchy* at the edge for IoT devices security using Dynamic Neural Networks that consume fewer edge resources. We further evaluated the robustness of these edge-based machine learning models against adversarial attacks by building *DDAS*, a GAN-based adversarial attack model.

3.2.2 ***iPrivacy*** - A Privacy preserving Service for IoT devices

The second component of our framework is a privacy service whose objective is to counter the side-channel attacks of network traffic-based privacy inference. We build two sub-components of this service: a Privacy Guard - a programmable and flexible network traffic shaping tool and *MirageNet* - a synthetic network traffic generating tool. While the *PrivacyGuard* tool can apply traffic shaping schemes such as padding, delay etc., to the network traffic for obfuscation from privacy attacks, *MirageNet* can generate fake network traffic flows mimicking the user-triggered device activities for countering the entropy of the side-channel attacks. We build *PrivacyGuard* using a programmable approach where the traffic shaping schemes can be selected dynamically based on certain policies, either user-defined or by an ML model, and applied on network traffic for obfuscation. For *MirageNet* , we propose to use a GAN-based model that can automatically understand the semantics and syntax of the network model of IoT devices through training and then generates similar network flows mimicking the flows of the original traffic without being detected synthetic. We build two tools here to have the flexibility of choosing traffic shaping or synthetic traffic based on context. For example, applying traffic shaping could be suitable for obfuscating real sensitive user activities, and synthetic traffic can give the illusion of user activities that do not exist. In the next chapters we discuss each of the components of the *iKnight* and *iPrivacy*.

CHAPTER 4

IKNIGHT - GUARDING IOT INFRASTRUCTURE USING GENERATIVE ADVERSARIAL NETWORKS

4.1 Introduction

To secure IoT devices, flexible network services with efficient and lightweight classifiers must be built. In this chapter, we build IoT classifier framework that using a semi-supervised GANs (SGAN) [20, 19], a recent extension to the GAN model, that has shown excellent results in image classification [20], online spam [27] reviews, and medical data [28] where the availability of labeled data is very limited. The SGAN model consists of a multi-class classifier that can perform automatic feature extraction with very limited labeled data. Moreover, in the training of SGAN, the classifier is exposed to the additional synthetic data points with noise that gives the classifier better generalization ability.

Several features of the SGAN model such as multi-class classification, automatic feature extraction, ability to be robust to noise, and capturing the hidden distribution of the of different classes [29] are aligned with the requirements and objectives of building a real-world IoT classifier. Moreover, the SGAN generator's ability to create synthetic data after training without needing access to training data will enable the IoT classifier to learn incrementally without any data storage requirements.

Motivated by the above observations, we developed IoT classifiers for real-world scenarios using SGAN. We summarize the contributions of this chapter as follows:

- We design and develop *iKnight*, a Semi-supervised GAN-based multistage multi-class classifier for classifying IoT devices for different scenarios such as known devices, unknown devices, malware, and anomaly with very few labeled data.

- We implement different real-world IoT classification features of *iKnight* such as continual learning and lightweight model deployment using generative replay and knowledge distillation techniques, respectively.
- We evaluate the features of *iKnight* using real IoT devices and malware network traffic datasets.
- We deploy *iKnight* on a real-world edge Hardware, NVIDIA Jetson Nano.

4.2 Background

4.2.1 Semi-supervised GAN

The amount of labeled data has a significant impact on the accuracy of classifiers in machine learning models. However, many real-world problems have less labeled data and would occur significant labeling efforts. Some researchers have proposed to utilize the unlabeled data using semi-supervised learning [30], which uses few labeled data points to identify the class of the majority of the unlabeled data points. With only a few labeled points, this approach approximately predicts the label of unlabeled data. Recently, some researchers have proposed semi-supervised learning using GANs, generally known as semi-supervised GAN (SGAN) [19, 20]. The SGAN has shown promising results for image classification problems that have sparsely labeled data [20]. From Figure 4 similar to GANs, the Semi-supervised GAN model consists of a generator and discriminator but has an additional classifier model that shares all the layers with the discriminator but have a different output layer [20]. The objective of the classifier is to classify the K classes of the training data, unlike the discriminator, which acts as a binary classifier and only predicts if the data is fake or real.

In SGAN, in addition to training the discriminator on unlabeled data, the classifier is trained on supervised labeled data. The Generator is given the loss of the discriminator. The rationale behind this is that the feature representations learned from the

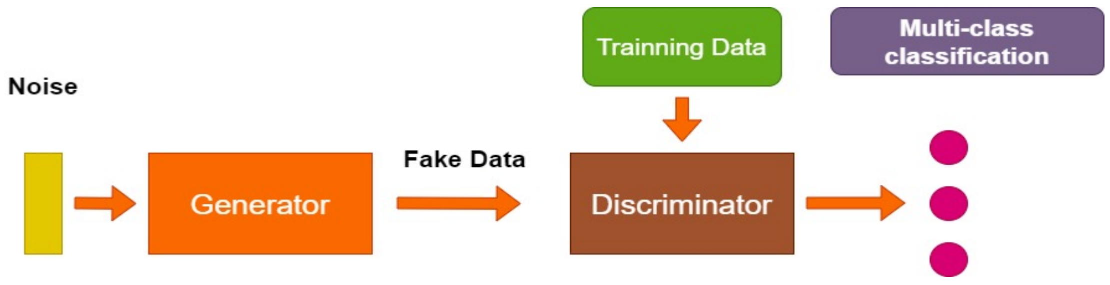


Figure 4: Semi supervised GAN (SGAN) Architecture

discriminator to identify the real data distribution improve classifier efficiency. Furthermore, the min-max game-based adversarial training will force the Generator to create synthetic samples that create additional data points to the classifier; thus, improving its efficiency [19]. In SGAN training, both the discriminator and classifier are exposed to real labeled data, real unlabeled data, and fake data from the Generator. This approach generalizes better than other semi-supervised methods that do not have access to additional synthetic data points during training. Moreover, Generator based fake data points include noise that makes the classifier robust to noise in real-world classification tasks.

4.3 Related Work

In recent years, there has been a significant interest in IoT device fingerprinting research [13, 31, 32] and activity inference analysis [12, 7]. While some works proposed supervised classifiers based on features such as device-dependent features for device identification [13, 33], others used time series based features [12] and flow level metadata [7] to detect activities of IoT devices. Recently, authors in [15] used CNN and RNN based deep learning classifier for network classification, but this technique is also based on supervised learning that is highly dependent on labeled data availability. Recently, researchers have proposed an unsupervised clustering approach for IoT device classification [34]. Also, in [35], authors utilize IoT-based raw network packets as their features for classifying IoT devices using autoencoders and bayesian modeling. We believe that multi-class classifiers are much more helpful for real-world scenarios given their ease of deployment, maintenance, and scalability.

Some researchers used GANs to generate synthetic packets [36, 37]. However, in this work, we use semi-supervised GANs for classification tasks. Very recent work has used a semi-supervised based approach towards classifying network applications belonging to QUIC, VPN, and Non-VPN datasets [38]. However, it is well-known that IoT-based network traffic is very different from traditional network traffic. In addition, although this recent work aimed at building classifiers for a limited set of applications with very unique characteristics (e.g., browsing, streaming), building a classifier for IoT network classification with many devices with similar network characteristics is far more challenging. Furthermore, IoT environments have several unique challenges such as concept drift, dynamic addition of devices, malware infections, and anomaly detection, which our work addresses.

In this chapter unlike the traditional approach of using flow-based metadata features we use a *2D flow-based encoding scheme* that is more relevant for IoT based classifiers since metadata features such as inter-arrival times can have adverse effects with the change in network conditions (e.g., limited bandwidth). In addition, we implement real world IoT classification features such as continuous learning feature using SGAN, which can continuously learn new device classes and deploy the lightweight model at the edge using knowledge distillation. Further, we design a software-defined network (SDN) enabled privacy-preserving service at the edge supported by our IoT classifier. Finally, we deploy and evaluate the performance of our SGAN based IoT classifier on NVIDIA Jetson Hardware.

4.4 Iknight Features

Our goal of designing and developing *iKnight* with a multistage and multiclass SGAN based implementation is to allow different interesting features that are essential to a real-world edge-based IoT classifier. In this section, we list and discuss the various features of *iKnight*.

Automatic feature extraction and limited labeled training data: Many IoT classifiers are based on supervised learning and require manual feature engineering [13, 39]. However, these techniques rely on the availability of labeled data, which is challenging given the privacy concerns associated with the IoT data collection process and the needed labeling efforts. Moreover, the network flows of different devices can have different network characteristics [7]. For example, the network flow of complex IoT devices such as *Withings* baby video monitor differs from the flow characteristics of simple IoT devices such as *Lifx* light bulb. Thus, the manual extraction of universal features that is efficient across all IoT devices is complex. In *iKnight*, the SGAN CNN classifier will automatically extract the hidden features during training. Moreover, the training of the SGAN model for each stage requires only 3%of labeled data. Therefore, *iKnight* supports both automated extraction of features and limited labeled training data.

Identifying unknown devices and zero-day attacks: In real edge networks, devices may dynamically join a network and create challenges for the network administrator in identifying the type of unknown devices (i.e., IoT, non-IoT, malware, or anomaly). In existing multiclass IoT classifier systems, unknown devices are typically classified as known classes that cause serious vulnerabilities. For example, a zero-day attack network traffic classified as a known device may allow the device to carry out attacks. Detecting the class of an unknown device is called novelty detection and is supported by the recent implementation of semi-supervised GAN[40]. Given the significance of this feature, *iKnight* utilizes the multistage architecture to realize this feature. More specifically, as described later, *iKnight* uses the multiclass classifier in stage 1 to detect any unknown device in which the multiclass classifier in stage 2 is, then, identify the type (i.e., IoT, non-IoT, malware) of the device. If stage 2 still identifies the device type as unknown, *iKnight* will mark the device flow as an anomaly.

Generalized model with robustness to noise: IoT network traffic suffers from concept drift where the network traffic signatures of the device may change over time due to firmware updates [34]. Moreover, bad network conditions such as high packet drops can create noise in the network flows and have an adverse effect on classification tasks. Furthermore, an over-fit model to training data can miss-classify a device-specific flow that has a slight deviation from training data. Therefore, a robust classifier should be well generalized to training data and effectively robust to noises, such as changes in payload and packet sizes or missing packets. In *iKnight*, the semi-supervised GAN model classifier is exposed to adversarial noise in the fake samples from the generator allowing the SGAN classifier to adjust its weights for any small deviation from the training data. Therefore, *iKnight* will be resistant to noises in both stages.

Flexible and continuous learning model: With the addition of new devices to the network, the model should update its knowledge by incremental training on new network data. However, this is challenging given the limited data storage space of edge systems. In addition, the preservation of IoT data for a long period of time would pose challenges to privacy. In *iKnight*, the semi-supervised GAN Generator learns the distribution of the training data during training enabling *iKnight* to use the generator to produce synthetic data similar to the training data without requiring any access to previous data. Moreover, GAN-based models can also learn incrementally without fully retraining by using the synthetic samples generated by the generator while learning new class data. Therefore, *iKnight* is flexible for continuing to learn new data.

In view of these features, iKnight can be used effectively and efficiently for device and device type identification. This device type discovery capability of iKnight can enable different network services on the edge that we discuss next.

4.5 Iknight Framework

In this chapter, we design and implement *iKnight*, an IoT network classifier for Edge-based systems. We discuss below the architecture and implementation of *iKnight*.

4.5.1 Architecture

Figure 5 shows an overview of the *iKnight* framework that consists of an Encoding Engine and Device Discovery Engine. In *iKnight*, the encoding engine encodes the new device's flow and passes the encoded flow to the device discovery engine. The device discovery engine uses multi-stage multi-class SGAN-classifiers trained using two different SGAN-GAN models to identify a new device class (device name) and its type (IoT vs. Non-IoT). In this chapter, we use SGAN-GAN model to refer to the combined models of SGAN-generator, SGAN-discriminator, and SGAN-classifier. We discuss below the design of the encoding engine and device discovery engine.

Encoding Engine: The *iKnight* encoding engine encodes the raw network flow packets of new devices using an encoding scheme. An encoding scheme is a process of mapping the raw network packets to a feature matrix that we define as a flow encoded matrix, which is sent to the device discovery engine for the device (name) and device type identification. The multi-stage multi-class SGAN model in the device discovery engine uses these raw encoded features to extract the hidden features and identify the device type automatically. An efficient encoding scheme is a significant factor in the efficiency of device type identification [41, 42]. In *iKnight*, we use two encoding schemes; *packet-only* and *packet-with-inter-arrival-time* schemes that we discuss later in the implementation subsection.

Device Discovery Engine: The *iKnight* Device Discovery Engine is a multi-stage classifier that can identify the device under different scenarios, such as identifying the device ID for a known device and the device type for an unknown device. We define a device as known if it is available to the SGAN-classifier during training and

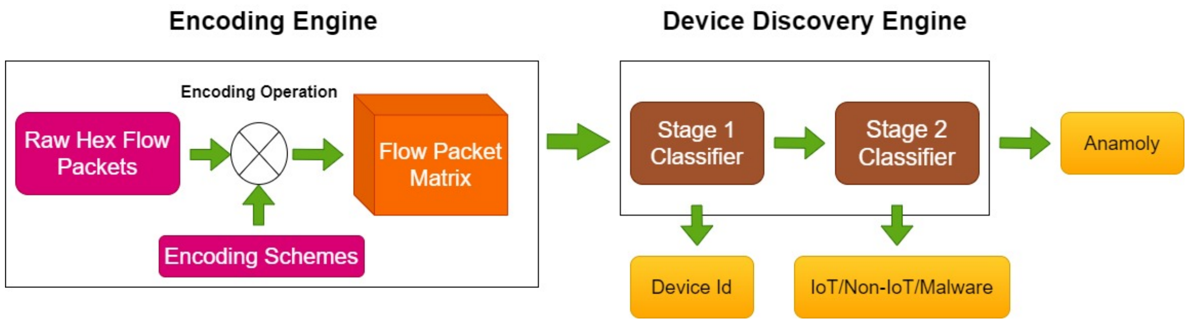


Figure 5: *iKnight* Architecture

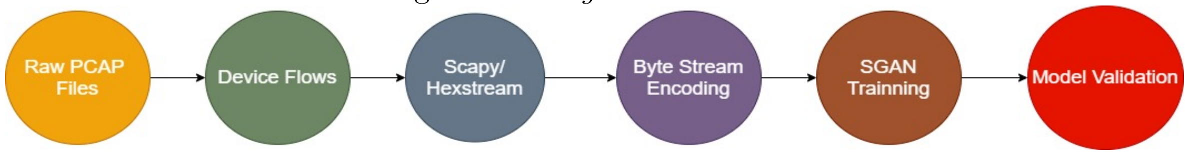


Figure 6: *iKnight* stage 1 and stage 2 model training pipeline

unknown otherwise. In *iKnight*, we take a two-stage approach for device discovery. In the first stage, *iKnight* uniquely identifies all known devices by classifying them into the corresponding classes. However, for any unknown device, *iKnight* classifies it as an unknown class, unlike other multi-class classifiers that will incorrectly identify an unknown class as one of the known classes. In the second stage, *iKnight* infers the device type of the unknown class as either an IoT, non-IoT, malware or unknown. However, we refer to the unknown class in stage #2 as an anomaly, given it does not fall into any known network traffic classes. This type of inference at stage #2 will help the network administrators to identify the types of unknown devices and, consequently, to adapt network policies. Some of these policies can give higher network priority to specific device types (IoT vs. Non-IoT), enforce any known anti-malware patches for malware, and issue alerts in case of an anomaly for immediate actions.

4.5.2 Implementation

We discuss below the datasets, training procedure, and the SGAN-GAN model architecture for both stages in detail.

Data sets and training We use the publicly available UNSW [33] and IoT-23

Table 1.: The list of IoT and non-IoT devices used in experiments

Device Name	Device Type
Smart Things	Hub
Amazon Echo	Speaker
iHome	Speaker
Triby Speaker	Speaker
Netatmo Welcome	Camera
TP-Link Day Night Cloud Camera	Camera
Samsung SmartCam	Camera
Dropcam	Camera
Insteon Camera	Camera
Withings Smart Baby Monitor	Camera
Nest Dropcam	Camera
Belkin Wemo Switch	Acutator
TP-Link Smart Plug	Acutator
Light Bulbs LiFX Smart Bulb	Acutator
NEST Protect Smoke Alarm	Sensor
Netatmo Weather Station	Sensor
Withings Smart Scale	Sensor
Blipcare Blood Pressure Meter	Sensor
Withings Aura smart Sleep	Sensor
PIX-Star Photo-frame	Digital Frame
Laptop	Non-IoT
Android Phone	Non-IoT
iPhone	Non-IoT
Samsumg Galaxy Tab	Non-IoT

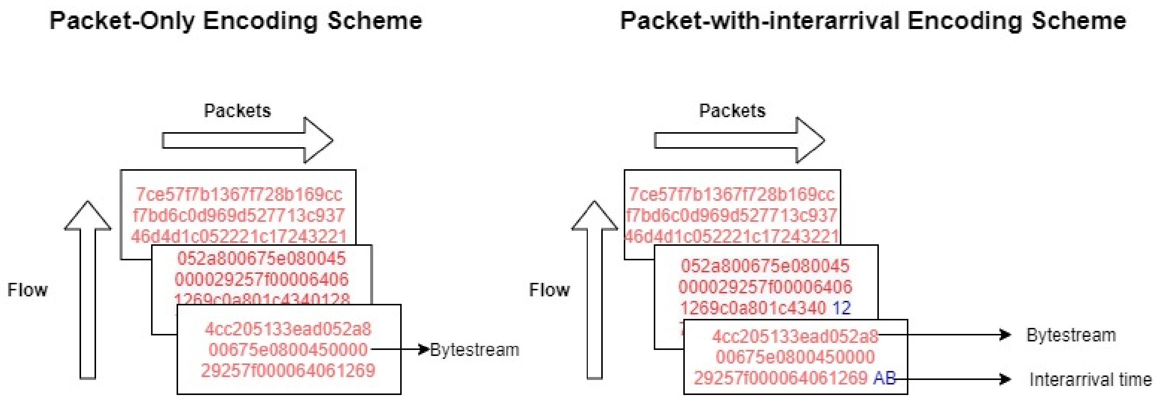


Figure 7: *iKnight* Encoding Scheme

datasets [43] to validate and evaluate our proposed device and device type identification approach. We use the network traffic traces from different classes of IoT devices and non-IoT devices from the UNSW dataset as shown in Table 1. The IoT-23 consists of labeled IoT malware network traffic flows from twenty different malware binary files (each different class) executed on a raspberry PI device in a controlled environment with flows labeled as either benign or malicious. We select 60,000 flows from the UNSW dataset and 10000 malicious flows from four different malware classes, Mirai, Trojan, Hakai, and Torii, for our experiments. For the stage #1 model, we train the model with twenty IoT devices, while for the stage #2 model, we use the combined network flows of twenty IoT devices, four Non-IoT devices, and four malware classes.

For both stages, stage #1 and stage #2, we train two different SGAN-GAN models and output two different SGAN-classifiers to identify device ID and device type identification. Figure 6 shows the SGAN-GAN model training pipeline that we repeat for each stage. We first extract flows from the raw pcap files using the pkt2flow tool[44], which splits the pcap files into individual flows. We then convert all the packets of a flow into raw byte stream using *scapy* tool[45]. For both models, we use raw TCP, and UDP flows for our training data. We remove the MAC layer header and device-dependent fields (source address, destination address, destination port, source port, and checksum) from IP and transport layer headers to make our training data network independent. We then train the SGAN-GAN model at each stage with two training data types: network-independent-headers-payload and only payload. We discuss the effect of the selection of each training data in the evaluation section. The training data at each stage consists of randomly sampled flows for all classes with balanced representation.

Figure 7 encoding engine showing the different encoding schemes *packet-only* and *packet-with-inter-arrival-time* encoding scheme to create the flow encoded matrix for each flow as the input to the SGAN-GAN model training. In the *packet-only* encoding scheme, we use the raw network byte-stream representation of the packets as features.

We first convert each byte-stream representation of the packet of the flow into their equivalent hexadecimal integer value. This hexadecimal flow array is then encoded as a three-dimensional image matrix, a flow encoded NumPy matrix with height, width, and depth. Each row of the flow encoded matrix consists of a packet hex stream arranged as a 2-dimensional array with a size of 56*56 (accommodates the maximum 1500 bytes of a MAC packet). The packet rows are ordered in the flow matrix as per their arrival sequence. This encoding scheme is similar to pixel intensity arrangement in a non-gray image matrix where each row is a 2D matrix representing the different channels of the image. The flow encoded matrix's depth is the number of packets in the flow, a configurable parameter, which we find as the first 5 packets based on experiments. If the flow has less than 5 packets, then the rest of the packets rows of the flow encoded matrix are appended with zeros, an approach similar to [35]. Similar to the *packet-only* encoding scheme, in *packet-with-inter-arrival-time* scheme, we use both the raw network byte stream and the packets' inter-arrival time. We arrange the 2D packets in the flow encoded matrix as rows similar to the *packet-only* scheme. However, we append an additional row at the end containing the inter-arrival times of the packets in their arrival sequence. Moreover, we convert the inter-arrival values to a hex decimal value similar to the packet data. We discuss the impact of these encoding schemes on device type inference in the evaluation section.

For both stage #1 and stage #2, we implement the SGAN-GAN model architecture described in [20] that consists of a SGAN-generator, SGAN-discriminator, and SGAN-classifier [20] for *iKnight*. The classifier shares the weights with the discriminator [20] and has a Softmax layer [46] used to classify the flows' device or device type classes label. For our SGAN-generator and SGAN-discriminator models, we adopt an architecture very similar to that of DCGAN guidelines [47]. Note that the approach of using DCGAN for SGAN is also very recently used by [38]. However, we design our model with layers and hyper-parameters optimized specifically for byte stream-based flow encoding scheme training data, which consists of inputs of a 56*56*n Numpy array, where n is the number of packets the flow. For SGAN-discriminator, SGAN-classifier, and SGAN-GAN model, we use Adam optimizer [48]. Below we discuss architectures for the SGAN-generator, SGAN-discriminator and SGAN-classifier, and hyper-parameters. The SGAN-generator model consists of 5 layers. The first two layers are fully connected layers that accept an input noise vector of size 100 and gives an output of shape 256*7*7. The output is reshaped to a vector of size 12544. The next hidden layers perform convolution transpose operations with shape 256*256. A convolution layer follows this with the tanh activation function. Each of the convolution transpose layers is followed by the Leaky ReLu activation function and batch normalization layers. The output of the generator is a matrix with dimensions 56*56*n.

The SGAN-classifier model consists of 6 layers. The first layer is a fully connected layer that accepts an input of 56*56*n from the generator. The next four hidden layers are convolution layers that perform the convolution operations having 2*2 filters of stride 3*3. The first and last hidden convolution layers have a shape of 128*128, while the rest of the layers have a shape of 256*256. Each of the three hidden layers is followed by the Leaky ReLu activation function and batch normalization layers. The output of the last hidden is reshaped and given to the fully connected dense layer. Finally, the SGAN-classifier output is a Softmax layer that provides the probability with each training class's predictions. For the SGAN-discriminator, we share all the layers with the SGAN-classifier except the output layer. The input to the SGAN-discriminator is the outputs from the SGAN-classifier last layer activation before Softmax. This SGAN-discriminator implementation is known to increase the accuracy of the classifier[20]. The SGAN-discriminator's output layer is a Sigmoid activation function classifying whether the input flow is real or fake.

In stage #1 SGAN-GAN model training, we split the dataset with 80% for training and 20% for testing. The SGAN-GAN model training consists of training the SGAN-generator, SGAN-discriminator, and SGAN-classifier with mini-batches of data in each epoch. Therefore, we select labeled flow samples for training the SGAN-classifier and unlabeled flows for training the SGAN-discriminator. Then, we train the SGAN-GAN model for multiple epochs, where each epoch consists of multiple batches of training data. We then validate the stage #1 SGAN-classifier model with the testing data. The SGAN-GAN model training process for stage #2 is similar to stage #1, except that we filter the flows of a specific device and use the filtered flows to infer the device type class during validation. For example, we filter out all the flows of Dropcam during training and then evaluate the trained SGAN-classifier model with these filtered flows to validate whether the trained model can infer the correct type of device as IoT. We repeat this process for all the device classes (IoT, Non-IoT, Malware) and report the average accuracy. This process effectively evaluates the ability of the SGAN-classifier to infer unknown devices, which is discussed in the evaluation.

Both stages of *iKnight* framework utilize the SGAN-classifier to achieve their objectives, known-device identification, and unknown device type -identification. The SGAN-generator model can also be utilized by *iKnight* for learning to classify new device classes or identify new device types that were not seen during the initial training process, which we discuss next.

4.6 Iknight Adaptability and Continuous Learning

IoT devices' network traffic can change over time due to firmware updates by the vendor or server API changes. This change of network traffic over time is called concept drift and can effect the IoT classifiers accuracy as they are dependent on network features such as flow level features (number of bytes sent or number of bytes received, etc.) or packet-level features (payload signature, etc.) derived from original network traffic. Moreover, to keep supporting the device or device type identification, the classifier must retrain with the updated traffic features. Furthermore, the IoT classifier would have to learn to classify any new device classes and their device types that are not possibly seen during training but added to the smart environment during the inference stage in an ad-hoc fashion. Therefore, the classifier should support class-incremental learning [49], a machine learning technique where the classifier can be trained on batches of classes incrementally. However, existing IoT classifiers [13, 15] do not support class-incremental training due to catastrophic forgetting [50], a phenomenon where the classifier accuracy will significantly drop after incrementally trained on new class data. The accuracy decreases because the model forgets the parameters (neural network weights) learned during prior training when retrained on the new class data.

To demonstrate the effect of catastrophic forgetting in an IoT classification scenario, we pick a subset of training data consisting of the 8 devices and split this subset into two datasets: Task #1 and Task #2, composed of 4 devices each. We then train a fully supervised CNN classifier with Task #1 data and then further retrain it on Task #2 data. We refer to this approach of training and then retraining on different tasks successively as class-incremental learning. Table 3 shows the accuracy of the CNN classifier and SGAN classifier for class-incremental learning. The accuracy of the CNN classifier reaches 100% after training it with Task #1 data, but the accuracy drops to 0% after retraining the classifier on Task #2 data. The rationale behind this significant drop is that during Task #2 training, the model updates its parameters learned during Task #1 training to classify the new classes of Task #2 thus forgetting the knowledge to classify Task #1 classes. A naive solution to address this challenge is to retrain the classifiers with Task #1 and Task #2 classes' combined data from scratch. However, this approach would require storing prior Task data, which is highly inefficient for storage reasons. Moreover, the IoT network data has very sensitive information [7] and keeping it on edge or remote servers for a long time could have severe security and privacy risks. Therefore addressing catastrophic forgetting, which is still an open and challenging area in the machine learning community, needs to be explored by the

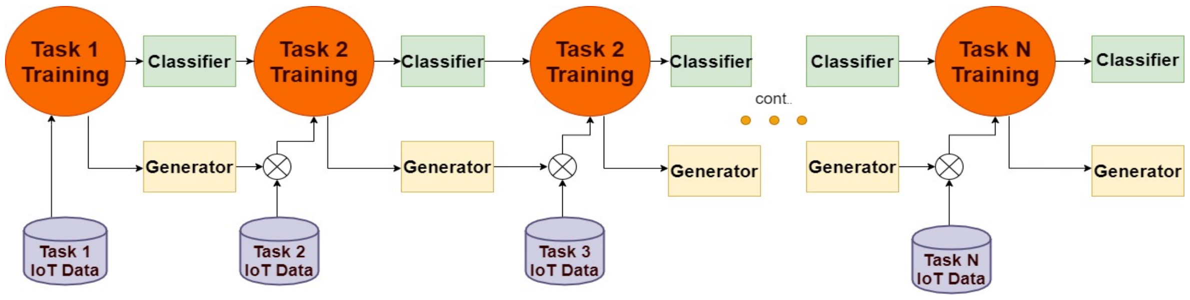


Figure 8: *iKnight*'s continuous learning across N number of Tasks

network community to support IoT-based class-incremental learning.

To address the catastrophic forgetting problem, we utilize the SGAN-generator's abilities to mimic training data to implement continuous learning in *iKnight*. We use the prior task generator to generate prior task synthetic samples and then augment it with the new task data to train the classifier incrementally as shown in Figure 8. Note that this approach is similar to [51] where the old generator(from prior training task) of ACGAN [52] is used to address catastrophic forgetting in continuous image generation tasks. However, here we implement the generative replay approach using the SGAN-generator for IoT-based network classification tasks. With this approach, edge systems do not have to store any old training data, thus not requiring additional storage. Moreover, this approach would require less computation time as it is not needed to train the model again from scratch.

In Algorithm 1, we show the continuous learning implementation for *iKnight* using SGAN. In Step #2, we first train the generator and classifier on the Task #1 device classes data, using traditional SGAN-GAN training discussed. In step #3, when the new device classes arrive, Task #2, we first generate Task #1 training data synthetically using the Task #1 SGAN-generator from Step #2 and augment them with the Task #2 training data (Step #3- #13). However, one of the challenges we face in an SGAN-generator is that it randomly generates the synthetic data without the class label, and for training the SGAN-GAN model on the current task, we need the generated

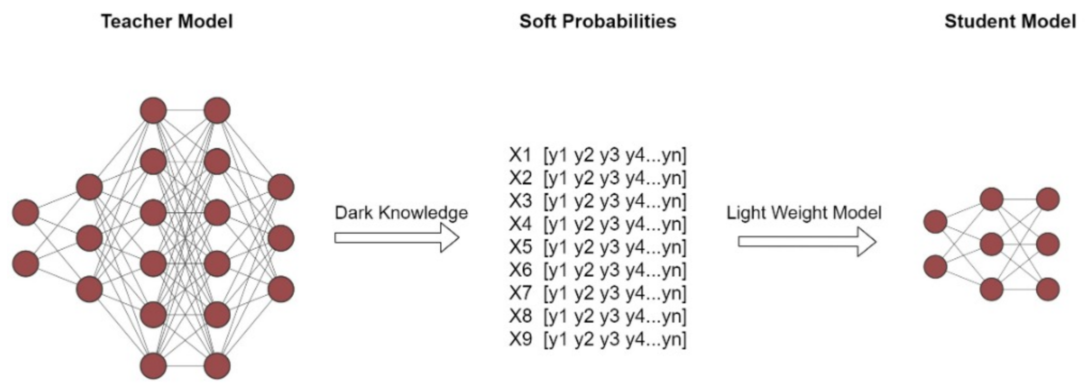


Figure 9: *iKnight* Lightweight transformation using Knowledge distillation

samples to be labeled. To address this, we use the prior task SGAN-classifier from Step #2 for labeling the generated samples. Moreover, we generate the samples until we collect a balanced dataset representing all the old training classes' classes equally. After collecting all the combined datasets of old (synthetic) and new training classes, in step 14, we perform the SGAN-GAN training. This process of incrementally training the SGAN-classifier can be repeated for the following N tasks, as shown in Figure 8 using the N-1 SGAN-classifier and N-1 SGAN-generator. We achieve good accuracy with this approach even without access to old training data (ground truth), as shown later in our evaluation section. Overall, *iKnight* can support many features required for edge-based IoT classifiers, as discussed in the above sections. Given these abilities, we discuss in the next section our strategy for optimizing *iKnight* for edge-based deployment on resource-constrained devices such as routers, switches, and edge servers which have strict application requirements such as high inference speed, less memory footprint, and low power utilization.

4.7 Iknight at the Edge

In recent years, there is an emergence in the deployment of deep learning-based applications on edge-based devices [53, 54, 55]. The main characteristics of these edge-

Algorithm 1 *iKnight* continuous learning using SGAN

```
1: procedure IKNGHT__CONTINOUS__LEARNING
2:     Perform SGAN training and get the GAN model
3:     while receive new tasks do
4:         Get the training samples and labels of current dataset
5:         while new tasks keep coming do
6:             calculate the number of samples to generate
7:             calculate the total number of old tasks till now
8:             while Generate samples using current generator till number of samples
   per class for all the old tasks are generated do
9:                 generate samples for each class in the task
10:                predict labels of the generated samples using prior SGAN classifier
11:                prepare the training data and corresponding class labels for this task
   and append to the new training data
12:            end while
13:        end while
14:        Perform SGAN training
15:    end while
16: end procedure
```

based applications are low latency, low memory, and less energy consumption. Given *iKnight* will be deployed on the edge networks we optimize the *iKnight* classifiers into very lightweight models that can achieve similar accuracy but with significantly less memory footprint and increased inference speed. This section discusses our approach for implementing the lightweight version of the *iKnight*.

Many well-known deep learning models such as RESNET [56] and VGG[57] have an intricate model design with many layers and parameters to learn. For example, RESNET, which is trained on CIFAR-10 image dataset [58] has a complicated model design with more than 100 layers and a total of 1.7M parameters. However, deploying such models on edge devices is challenging due to resource constraints. Some researchers have recently proposed a knowledge distillation technique to [59, 60] to train a lightweight model by distilling it with the knowledge of an already trained complex model. In this approach, a teacher model is first trained on the training data, which learns the parameters (network weights) required for classification tasks. During the training, the Softmax layer learns each class's probability distribution which can be smoothed to form Softargets or training labels for the student model. Smoothing the probabilities can help discover other information about the training data, such as the proximity of different class features to the predicted class, and helps the student model to converge quicker. Therefore, unlike the traditional method of training the model with training samples and corresponding labels, the student model is trained with training samples and the teacher model's predicted soft probabilities as shown in Figure 9. Note that softening the probabilities is done through a hyper-parameter called Temperature; the higher this value is, the smoother are the probabilities.

We implement the knowledge distillation approach using the *iKnight* Stage #1 SGAN-classifier as a teacher and a new lightweight CNN model as a student. We take the trained stage 1 SGAN-classifier as a teacher model with 4 convolutional layers, using SGAN to create soft probabilities or soft targets for the student model. We then train a very small CNN model with only one convolutional layer, which has fewer layers than the teacher model and uses Soft probabilities-based training data. The student model achieved an accuracy of 91% very close to our Stage #1 teacher model but with fewer layers as shown in our evaluation section later. In Our experiments, we evaluated different student model architectures and found, the CNN model with one convolutional layer gave the best accuracy. With this approach, we optimized the *iKnight* Stage #1 classifier for edge-based scenarios with less inference time, less memory footprint, and less power consumption, which we discuss in our evaluation section. One of the other advantages of this approach is that we can train our Stage #1 and Stage #2 SGAN-classifiers anywhere on cloud or edge servers and then deploy lightweight models much easily with the help of knowledge distillation on the edge devices.

From sections 4.5 and 4.6, we can see that *iKnight* is equipped with different features that can discover device types, both known and unknown, and can continuously learn new device types. Moreover, the knowledge distillation approach *iKnight* can be transformed into a much lightweight model and can effectively support many new network services at the edge. In the next section, we discuss the potential of *iKnight* to provide such services.

4.8 Evaluation

4.8.1 Experiment setup

We implement *iKnight* Device Discovery Engine using Keras [61] and python, and train it on a GPU enabled machine with installed and 32 GB of memory. In the below sections, we evaluate *iKnight* for the different features of a real-world classifier.

4.8.2 Impact of training data and encoding schemes

Figure 10 shows how the accuracy of the stage #1 classifier improves with the number of labeled samples. Therefore, the device type classification depends on the

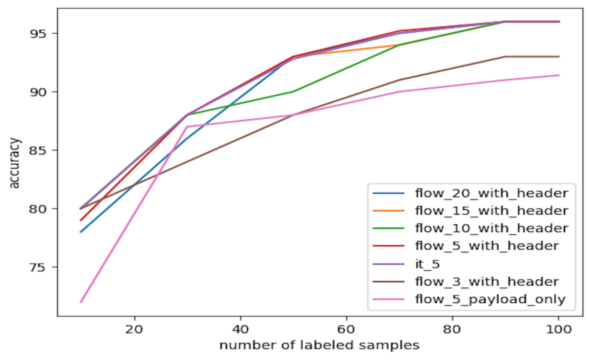


Figure 10: The accuracy of stage #1 classifier in *iKnight*'s Device Discovery Engine

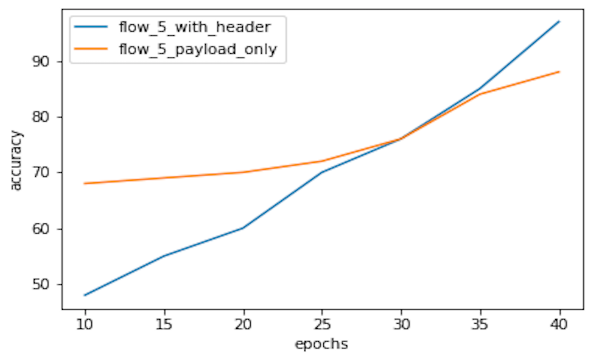


Figure 11: The accuracy of stage #2 classifier in classifying an unknown device

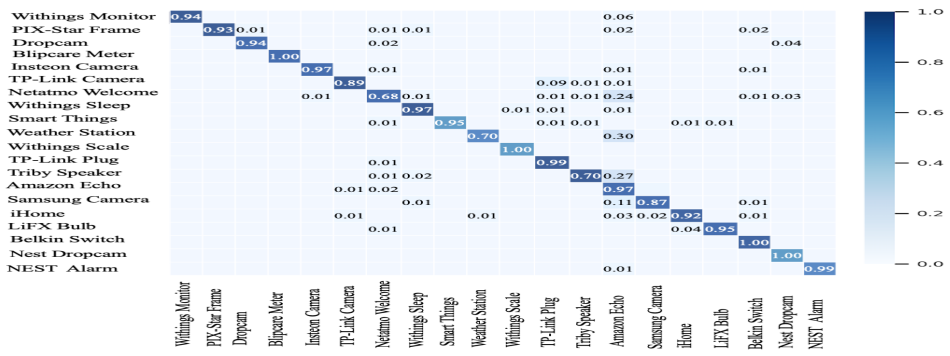


Figure 12: The confusion matrix for *iKnight* stage #1 classifier with packets-with-payload-only

number of labeled samples, as discussed earlier. In our experiments, for both stage #1 and stage #2 classifiers, we achieve the highest accuracy with only 3% labeled flows of each device. Therefore, the best configuration of *iKnight* can achieve high accuracy with very few labeled data. We show the confusion matrix for stage 1 classifier using packets-with-payload-only encoding scheme in Figure 12.

In addition, Figure 10 also shows the increase in the accuracy of stage #1 classifier from 92% to 96% when the training data selection is changed from packets-with-payload-only to packets-with-network-independent-header. Therefore, the network-independent-header fields for IoT devices significantly improve the IoT device classification due to its features that help the classifier to uniquely identify each of the device classes more than packets-with-payload-only. Similarly, Figure 11 shows that the stage #2 classifier's accuracy significantly improves by 9% with the change of the

Table 2.: The accuracy of *iKnight* stage #1 classifier using packets-with-payload-only encoding scheme across twenty IoT devices shown in Table 1

Classification Method	**Accuracy**
Kmeans	0.01
Kmeans + PCA	0.03
KNN	0.85
Decision Tree	0.86
Random Forest	0.88
CNN	0.90
SGAN	0.92

encoding scheme from packets-with-payload-only to packets-with-network-independent-header while identifying the device type of an unknown device (i.e., Dropcam). Therefore, the network-independent-header improves the unknown device type classification.Moreover, recently some researchers observe that IoT device types use specific device based network configurations such as using PUSH FLAG for faster network data processing on IoT client devices [8].

Moreover, we observe from Figure 10 that the system's accuracy increases by 3% with an increase in the number of flow packets from three to five. However, a further increase in the number of packets does not affect the classifier's accuracy. Additionally, the encoding scheme flow-with-inter-arrival-time with five packets also does not increase the accuracy. The rationale behind this could be that the weights of the SGAN classifier have learned the best possible parameters from the raw encoded packet data. The additional meta-data information (inter-arrival time) does not improve their ability to increase the classifier's overall feature space representation. However, we note that a better choice of encoding design, such as the location and the representation of inter-arrival times inside the flow encoding matrix, can improve the system's accuracy. However, finding the optimal system encoding scheme for *iKnight* is outside the scope of this chapter. Therefore, the best configuration of *iKnight* is the first five packets of the flow with network-independent-header and the encoding scheme flow-only.

4.8.3 Performance comparison to other classifiers

One of the advantages of SGAN is its ability to achieve high accuracy compared to supervised machine learning algorithms with less labeled data. Therefore, we compare *iKnight* with the most popular classification algorithms used for IoT network classification tasks. We compare the SGAN stage #1 classifier using packets-with-payload-only encoding scheme with both supervised and unsupervised classifiers as shown in Table 2. Each of the supervised classifiers was trained with 3% labeled data, and each of the unsupervised classifiers was trained with all the unlabeled data. Finally, the SGAN classifier was trained with both the 3% labeled and the unlabeled data. We use the same packets-with-payload-only encoding scheme for all the classifiers. Table 2 shows that SGAN, which was trained with the same number of labeled flows, outperforms all other supervised classification algorithms. Moreover, SGAN also exceeds the unsupervised K means algorithm, where both SGAN and K means were trained using the same number of unlabelled training samples. While SGAN extensively performs well over all other algorithms, we have around 2% increase in accuracy over CNN. We believe this can be further enhanced by a better choice of encoding schemes, which can help SGAN extract the training data's distribution very well. For example, a combination of flow-level metadata and flow-byte stream encoding can help GANs model with a lot of information during training and, consequently, build a better latent space representing the training data. Furthermore, accuracy is not the only metric to evaluate SGAN's potential for IoT classification, as SGAN outperforms CNN in many other scenarios such as noise and continual learning tasks, which we discuss next.

4.8.4 Robustness to noise in IoT device Traffic

This section evaluates the performance of *iKnight* under two noise-based scenarios: changes in payload and change in packet sizes. For the change in packet payload scenario, we pick the random locations of each packet's payload in a flow and change it with random hex values. Similarly, we append random hex values at the end of the packets to change the packet sizes scenario. For both scenarios, we introduce the noise only in the testing data.

Figure 13, stage 1 classifier, we can see that SGAN-with-payload-only is more robust to change in payload-based noise than CNN-with-payload-only. Therefore SGAN is a better choice for building robust classification models for IoT classification tasks with less labeled data. Moreover, Figure 13 SGAN- packets-with-network-independent-headers has only a small change in accuracy 10% for a noise of change of the percentage of the payload of 40%. During training stage 1, the classifier has additional noise training data from the SGAN generator, and therefore *iKnight* is robust to reasonable changes in payloads of the network traffic. However, for noise up to 40% with SGAN-payload-only, there is a decrease of 16% in accuracy. Therefore, a choice of independent network headers is a more reliable approach to build robust IoT classifiers. In our experiments, we observe an increase in the payload's size by 40 % for the SGAN-packets-with-network-independent-headers, the accuracy decreases by 20%, and for SGAN-packets-with-payload the accuracy decreases by 24%. Given in real-world scenarios, such large changes to payload are rare since the already deployed IoT devices' operational requirements generally can have small changes in the API methods that introduce small changes to the payload.

4.8.5 Unknown device type and anomaly detection performance

In this section, we evaluate *iKnight* for the different unknown device type and anomaly detection scenarios.

For evaluating *iKnight* stage #2, unknown device type identification scenario, we randomly choose five IoT devices, three Non-IoT devices, and all the four Malware classes. For each of the unknown devices, we first filter out the corresponding flows from the training data and validate the trained model with the filtered out flows of the

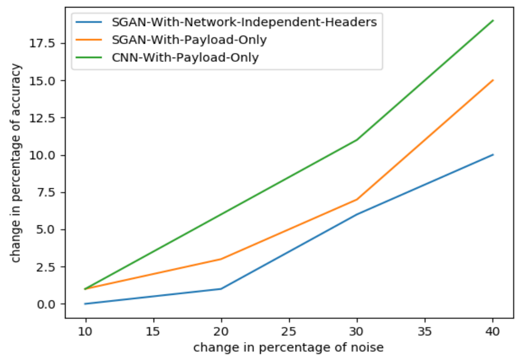


Figure 13: The change in *iKnight*'s accuracy of stage 1 classifier with different noise levels

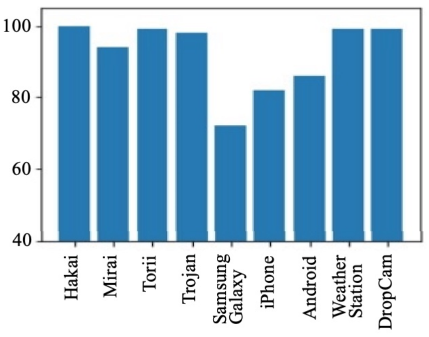


Figure 14: The accuracy of *iKnight* stage #2 classifier of unknown devices using packets-with-network-independent-header for different type classes (i.e., IoT, Non-IoT, Malware)

unknown device. We repeat this process for all the selected devices. Figure 14 shows the stage #2 device type classification accuracy using packets-with-network-independent-header encoding scheme for different classes of unknown devices. The accuracy of the unknown IoT and malware device types is higher than that of unknown Non-IoT device types. We believe this could be due to the few number of Non-IoT devices in the training data in which a better representation of each device type class in the training data can further improve the classification accuracy of the stage #2 classifier. The average accuracy for each of the device type classes for the unknown scenario is IoT device type with 94%, Non-IoT with 80%, and the Malware with 97%. The overall unknown accuracy for different classes of device types is 90%.

Figure 11 shows the Stage #2 classifier's ability in identifying the type of an unknown device (i.e., Dropcam) as an IoT device with 97% accuracy. The classifier's accuracy is higher when the model is trained with packets-with-network-independent-header than when trained with packets-with-payload-only. This shows that *iKnight* can effectively capture the hidden feature space distribution for each of the IoT, Non-IoT, and Malware known device types. Moreover, this enables *iKnight* to have a better-generalized feature space representation of each device type class, which improves its ability to identify the type of unknown devices.Therefore, *iKnight* is capable of identifying the class types of unknown devices and known devices.

Semi-supervised GANs have recently shown good performance in identifying unknown classes as unknown (anomaly)[40], also known as novelty detection. This SGAN capability helps the Stage #1 classifier to identify any device that is not part of the training data as unknown, and also benefits the Stage #2 classifier to identify the type of unknown device as IoT, Non-IoT, Malware, or any anomaly (i.e., novelty detection). As future work, we are planning to run more experiments to evaluate SGAN capability in anomaly detection.

4.8.6 Continual Learning with new classification tasks

In this section, we evaluate the continuous learning abilities of *iKnight*. We create three training scenarios wherein each scenario; we train the classifier on Task #1 and Task #2 data incrementally. Task #1 data represents 4 IoT devices classes, and Task #2 represents another set of 4 IoT devices. We then perform continual learning on the SGAN model for three scenarios. In the first scenario, without-old-tasks-data, the classifier is first trained on Task #1 and is then incrementally trained on Task #2 but without Task #1 data. In scenario 2, with-old-task-data, the classifier is trained on Task #2 incrementally with both Task #1 and Task #2 data. Finally, in scenario 3, with-generator-replay, the classifier is trained incrementally on Task #2 with Task #2 data and synthetically generated Task #1 data from Task #1 SGAN generator, as discussed in Algorithm 1.

From Table 3, we can see that for scenario without-old-tasks-data, both CNN and SGAN models have a tremendous accuracy drop from 100 to 0 respectively when incrementally trained on Task #2 data. However, on a change of scenario to with-old-task-data, the SGAN performs very well since the model learns to generalize the features for all device classes across Task #1 and Task #2 given it is trained on the ground truth data of both tasks together. However, with-generator-replay, the accuracy drops to 91, given generated samples will not precisely be the same as the ground truth Task #1 data. However, the Task #1 generator synthetic samples are still a good representation of the Task #1 ground truth data data, making the classifier achieve very high accuracy compared to the scenario without-old-tasks-data. Therefore, a continuous learning approach using the semi-supervised GAN generator can be an effective solution for IoT-based continual learning network tasks.

Model	**Method**	**Training Task**	**# Classes in Training data**	**Task 1**	**Task 2**
CNN	without-old-tasks-data	T1	4 (T1)	100	-
SGAN	without-old-tasks-data	T1	4 (T1)	100	-
CNN	without-old-tasks-data	T2	4 (T2)	0	94
SGAN	without-old-tasks-data	T2	4 (T2)	0	97
SGAN	with-old-task-data	T2	4 (T2) + 4 (T1)	100	97
SGAN	with generator	T2	4 (T1) + 4 (T2 - Generated)	90	97

Table 3.: Accuracy across continual learning tasks for IoT based classifier using SGAN and CNN models

4.8.7 Deployment on Edge Hardware

As discussed earlier, for optimizing *iKnight* for edge-based deployments, we use a knowledge distillation approach. In our experiments, we achieved an accuracy of 90% for the student model with only one convolution layer and a very small drop in accuracy compared to the teacher model (Stage #1 model with payload-only encoding scheme) with 92% accuracy. The ability to achieve higher accuracy with very small models can have significant edge hardware performance, which we discuss next.

Recently, different hardware platforms such as Raspberry PI [62], Google TPU board [63], Intel neural compute stick [64] and NVIDIA jetson nano [65] are being manufactured to support edge-based deep learning-based applications. To evaluate the deployment of *iKnight* on edge hardware, NVIDIA Jetson Nano [65] was selected. The NVIDIA Jetson Nano is a hardware device that has shown better performance for deep-learning-based applications [66]. The hardware configuration of Nano consists of a 32 GB SD card and 4 GB memory. We then installed the Tensor Flow [67] and other python dependencies [68] to run the deep learning based models on Nano and then convert the student and teacher models into a Tensor graph using Tensor RT library [69] as shown in Figure 15. The Tensor RT graph version of the student and teacher models are the optimized model versions tuned for the Nano hardware architecture. We then performed inference on both Stage #1 and Stage #2 models individually for predicting a sample 2000 times and measured the average performance metric using the Jetson stats tool[70]. Table 4 shows that the Nano student model outperforms the Nano Teacher version in different performance metrics. The student model has an increase of 41% in memory footprint compared to when the hardware was in an idle state; this is significantly lower than the increase in the teacher model's memory footprint with 81%. Similarly, while the student model had only an increase of 28% in power consumption, the teacher model had a significant increase of 147% when compared to the idle state. Finally, the student model's average inference time is 46% less than that of the teacher model. Therefore, a student model can achieve high-performance objectives for edge-based devices with less drop of in accuracy.

Given these results, we can conclude that training the models using a semi-supervised GAN to achieve different IoT classifiers' objectives and then utilizing the SGAN models to transfer the knowledge to the lightweight version for edge deployments complement each other well.

Model	**Accuracy**	**Change in % Memory(MB)**	**Change in % Power consumption(mw)**	**Inference Time (ms)**
Teacher Tensor RT	0.92	79.93	146.97	0.0116
Student Tensor RT	0.90	41.4	28.46	0.0062

Table 4.: Average Performance of *iKnight* stage 1 Student and Teacher Models deployed on NVIDIA Jetson Nano Hardware

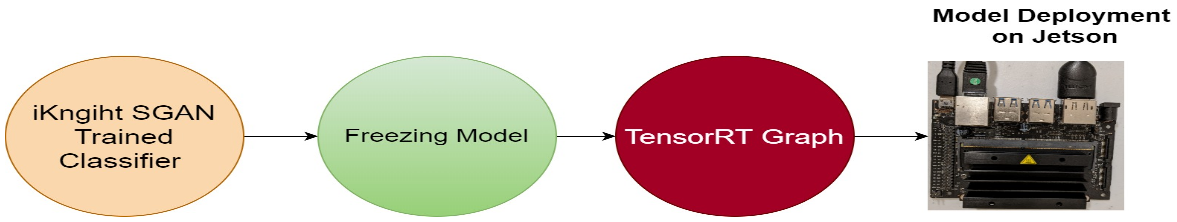


Figure 15: *iKnight* deployment on Edge Hardware, NVIDIA Jetson Nano

4.9 Conclusion

In this chapter, we evaluated and implemented the features of SGAN based IoT classification system. From the results, we can conclude that a semi-supervised based GAN approach has several features that benefit the design of an IoT classifier. Moreover, with our multi-class multi-stage classification approach, *iKnight* has more potential to identify both known devices and unknown device type.

We were also able to demonstrate the ability of *iKnight* to continuously learn IoT classification tasks when new devices are added or their firmware is updated. However, one of the challenges we faced in continual learning tasks is the quality of the generated samples over the number of tasks. With the increase in the number of tasks, the SGAN has to perform better in generating samples representing the old ground truth classes data. However, this is a going research problem by itself. We plan to further improve the continual learning abilities of *iKnight* by increasing the generator quality[71]. Moreover, we plan to combine different combinations of continual learning like elastic weight consolidation [50] and generative methods [51] together on GAN-based classifiers to better assist *iKnight* in classification and novelty detection tasks. Moreover, as discussed in our evaluation sections, training *iKnight* with datasets representing a more extensive set of device types classes that can further improve the reliability and accuracy of the *iKnight* to support different types of network services.

Next, in chapter 5, we would like to evaluate a new approach for deploying machine learning models for edge devices using dynamic neural networks. Moreover, given some recent work has shown that adversarial attacks can be used to modify malicious network traffic to be classified as benign by perturbing certain features of the malicious traffic [72, 73, 74] we explore the adversarial attacks and defenses for edge based models to make it more robust in chapter 6.

CHAPTER 5

IBRANCHY: AN ACCELERATED EDGE INFERENCE PLATFORM FOR IOT DEVICES

5.1 Introduction

The performance of DNN models when running on edge-based IoT devices is significantly impacted by the limitations of the device resources, which will reflect on the performance of these devices. Therefore, it is highly desirable to develop techniques to optimally accelerate the inference computations of DNN models in order to enable real-time applications and conserve energy for edge devices/IoT.

Very recently, different types of DNNs referred to as Dynamic DNNs (D^2NN) have been proposed [75] to provide low-latency and power-saving on IoT/edge devices. In contrast to traditional DNNs, dynamic DNNs are capable of performing conditional computations and selectively activate just sections of the network model, whereas traditional DNNs use the entire network model in the computation even when only a certain portion of the network is sufficient to make the inference. For example, BranchyNet [76], one of the popular D^2NN models, terminates its computation and infers early if the earlier layers of the network have sufficient confidence without requiring all of the subsequent layers to participate in computation, thus reducing inference latency and power consumption. The typical architecture of BranchyNet model consists of a network with multiple layers and different branches. Every branch of the network is followed a classification output component known as exit point. During inference, the early termination at a branch happens only if the exit point has adequate confidence to make the inference. Recent research has demonstrated that for the vast majority of inputs, the model will exit at the early exit point of the network during inference without requiring computation from the rest of the network [76]. These D^2NN models, which offer characteristics like on-demand computing, hardware adaptability, and fewer resource constraints, are therefore gaining popularity for constructing and developing high-performance IoT/edge applications [77, 78].

Motivated by the above observations, in this chapter, we design and develop a dynamic DNN (D^2NN) IoT classifier based on BranchyNet as a model classifier to classify actual IoT and non-IoT devices and evaluate the model with edge-based constraints such as inference time and power consumption. We summarize the contributions of this chapter as follows:

- We design and develop *iBranchy*, a dynamic early exit multi-class classifier for classifying IoT devices based on their network traffic.
- We evaluate the features of *iBranchy* using both real IoT and non-IoT devices.

5.2 Background

5.2.1 Static Compression vs Dynamic DNNS (D^2NN)

In recent years there has been significant research over accelerating machine learning models for edge deployments using different approaches [79] such as compression [80, 81] and knowledge distillation [59]. In compression-based techniques, the original DNN network may be pruned in different ways to remove any insignificant parts of the network. For example, in [80] authors compress the network using quantization method while authors in [81] compress the convolutional layers through a redundancy approach. However, in knowledge distillation technique [59], instead of compressing the neurons or weights of an existing complex DNN network model, a new lightweight network called student model is designed with very few layers and is trained to learn from the knowledge representations outputs of the pre-trained complex teacher model. This method has shown to be very effective with high performance for resource constraints and an insignificant drop in accuracy.

One of the major disadvantages of the above accelerated methods based on compression or distillation is that they are static with respect to computation and inference time as all the components of the DNN network will participate in inference phase computation irrespective of the type of input or environmental conditions. In contrast to static networks, dynamic networks D^2NN can change their internal structure or parameters during the inference phase, giving them greater flexibility and better adaptability to the underlying use case [75]. The D^2NN models accomplish this dynamic flexibility and adaptability through a conditional computation architecture that allows them to only selectively activate particular sections of their network based on context, such as input data or environmental conditions. Early-Exit-based models [76, 75] are one of the most popular categories of the D^2NN networks that have a multi-exit design where an exit is an early inference point attached to selected components of the network and can be conditionally activated based on the complexity of the input data. Therefore, in this chapter we use BranchyNet [76] a very well adapted Early-Exit-based D^2NN model that is gaining popularity for edge scenarios to implement our D^2NN based IoT network classifier.

5.2.2 BranchyNet

BranchyNet is an Early-Exit-based D^2NN model that supports an early inference of certain input samples using multi-branch and multi-exit design. Like a traditional DNN classifier, the BranchyNet network architecture consists of a multi-layer network followed by a softmax layer for output predictions. However, in addition to the main network, a small network called branches are added to the outputs of different layers of the main network. These branches, similar to the main network are also followed by softmax layer. The outputs from the different softmax of the different branches and the main network are called as exits. The multi-exit approach implemented in BranchyNet is based on the observation that the earlier layers of the network can perform inference for most of the input samples, thus allowing most of the inputs to exit early and thus reducing the overall network computation and reducing the average runtime and power computation.

In BranchyNet training a softmax cross entropy loss function is used for minimizing the network misclassification rate similar to traditional DNNs. However, the overall loss of the network is calculated using a weighted loss function consisting of losses at different exits of the network. The choice of the weights for each exit-specific loss is a hyper-parameter and impacts the model performance. During the inference phase, an input sample exits the network only if it is predicted with a confidence that is within the confidence threshold assigned to that particular exit. More specifically, BranchyNet uses an entropy-based confidence threshold where a sample exits from a particular exit only if it was predicted with entropy less than the threshold assigned for that particular exit. If the entropy of the input sample is larger than the given threshold, the sample is sent to the next exit for inference and the process continues till the sample reaches the final exit at the end of the main network.

The choice of thresholds at different exits is a run-time hyper-parameter that impacts model inference phase performance. The lesser threshold value at an exit will ensure the samples predicted with high entropy to be pushed to later exits, thus improving the model's accuracy at the cost of early inference. Two approaches are proposed in BranchyNet for threshold selection, in the first approach, different threshold values could be tested in an iterative method and finally choosing the best configuration based on user requirements such as higher accuracy with an increase in inference time or lower accuracy with a decrease in inference time. The second approach is to use a neural network-based approach that can automatically fine-tune the threshold values. In this chapter, we implement the first approach to pick the threshold values based on performance requirements.

5.3 Related Work

Given the significance of IoT device classification, some of the recent IoT network classifiers [33, 34] utilize either probabilistic model [33] or traditional machine learning based unsupervised model [34] for classifying network based IoT devices, but no deep learning-based solutions. However, in this work, given the growing popularity of deep learning-based classifiers that provide the features such as automatic feature extraction and better accuracy from raw network data [35, 41, 15], we build D^2NN based model for IoT classification that augments traditional DNN based features with additional features that can support edge-based resource constraints. For example, existing IoT classifiers that use different deep learning based approaches such as CNN-RNN [15], autoencoders-bayesian modeling [35] and semi-supervised GAN [41] require all of their DNN network entitiess to participate in computation for realizing their system. In contrast, our approach can conditionally activate only selected sections of the DNN network to save energy and inference time.

Other D^2NN related advances include developing a partition method [78] with BranchyNet and distributed DNN techniques [82] across the cloud and the edge devices for better performance and faster response times. Similarly, FlexDNN [77] - a Early-Exit based model accelerates video analytics on resource-constrained devices. However, our work significantly differs from other edge-based D^2NN models as we design, implement, and evaluate *iBranchy* for IoT-based network devices which is more challenging due to the fact that network data is not simple as traditional image or set of frames based video data. Furthermore, we perform a data transformation for the training data based on the encoding scheme [41] to implement BranchyNet [42] for IoT devices effectively. Very recently, in [83] authors use a context-based approach that selects the most appropriate anomaly detection model from the hierarchy of models deployed at the edge, cloud, and device to meet edge-based resource constraints, and it is implemented using sensor and power consumption datasets. However, unlike their work which use distributed models,

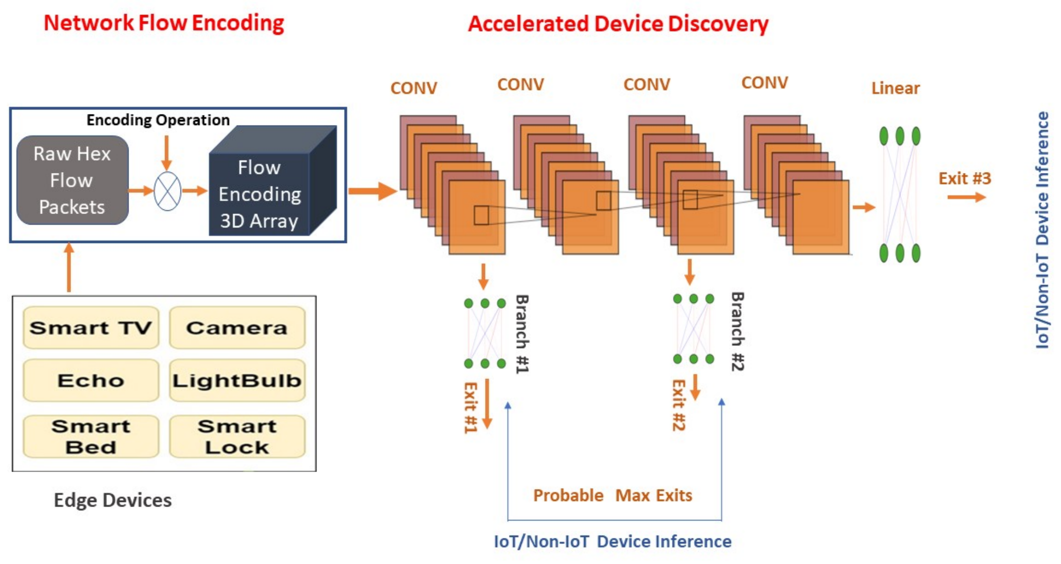


Figure 16: *iBranchy* Accelerated Edge Classifier

we implement a single model classifier based on BranchyNet that can uniquely identify IoT/Non-IoT devices using IoT network traffic.

5.4 *iBranchy* Framework

In this chapter, we design and implement *iBranchy*, an IoT network classifier for edge-based systems. Figure 16 shows an overview of the *iBranchy* framework that consists of two components a network flow encoding component and an accelerated device discovery component. The new devices' network flows are encoded through the network encoding component and sent to the accelerated device discovery component, a BranchyNet based Early-Exit multi-class classifier, to identify the new device class (device name). We discuss below implementation of each of these components in detail.

Network Flow Encoding: The network flow encoding component encodes the network flows of the devices by mapping the raw network packets of a flow into a three-dimensional array and is same that we used for*iknight* classifiers discussed in chapter 4 . This encoded flow array is given as an input to the accelerated device discovery component to extract the hidden features and uniquely identify the device. This flow-based encoding scheme using raw packet streams is a more efficient solution for BranchyNet based model, which uses CNN layers and can extract spatial features of the network byte data much efficiently.

Accelerated Device Discovery: The *iBranchy* accelerated device discovery component uniquely identifies a device into its corresponding class. However, unlike other popular DNN based IoT classifiers where all the layers participate in computation [15, 35, 41], *iBranchy* may exit at different stages of the model with probable very early exits that can significantly decrease the inference time and consume less energy. To design the *iBranchy* model we first design a baseline DNN IoT classifier and then, using the baseline DNN, we build an IoT based BranchyNet version with newly added branches where a branch is a set of layers or a very small DNN with early exit point as discussed in section 5.2.

Figure 16 shows the overall architecture of the *iBranchy* model, which consists of multiple convolutional layers followed by a linear layer with final exit and two branch exits. In *iBranchy*, we try different design configurations and choose the early exits at Branch #1 and Branch #2 at the CONV layer #1 and CONV layer #3, respectively. At the inference stage, if the entropy for a branch of a testing sample is less than the runtime threshold, the inference can be made early without further computation from subsequent layers. We chose a lightweight design for the branches with linear layers to add less overhead at run time for edge deployments but that could be further optimized. In *iBranchy*, the probability of a network layer to participate in the inference computation is dependent on the threshold values set for different exits. Therefore, at run time, based on the environment requirements such as battery power (low or high) or Wi-Fi signal strength *iBranchy* can be configured dynamically to exit more samples at early layers to save edge resources. However, choosing very low threshold values can significantly impact the inference accuracy and therefore needs to be optimized based

Table 5.: The list of IoT and non-IoT devices used in *iBranchy* experiments and also were used in *iknight* experiments

Device Name	**Device Type**
Smart Things	Hub
Amazon Echo	Speaker
iHome	Speaker
Triby Speaker	Speaker
Netatmo Welcome	Camera
TP-Link Day Night Cloud Camera	Camera
Samsung SmartCam	Camera
Dropcam	Camera
Insteon Camera	Camera
Withings Smart Baby Monitor	Camera
Nest Dropcam	Camera
Belkin Wemo Switch	Acutator
TP-Link Smart Plug	Acutator
Light Bulbs LiFX Smart Bulb	Acutator
NEST Protect Smoke Alarm	Sensor
Netatmo Weather Station	Sensor
Withings Smart Scale	Sensor
Blipcare Blood Pressure Meter	Sensor
Withings Aura Smart Sleep	Sensor
Belkin Wemo Motion Sensor	Sensor
PIX-Star Photo-frame	Digital Frame
Laptop	Non-IoT
Macbook	Non-IoT
iPhone	Non-IoT
Samsumg Galaxy Tab	Non-IoT

on the operating constraints and is out of the scope of this work. In the next section, we evaluate *iBranchy* for edge based deployment requirements.

5.5 Evaluation

In our experiments, we extract 56,000 flows from the combined network flows of twenty-one IoT devices and four non-IoT devices as shown in Table 5 from the publicly available UNSW [33] dataset to evaluate and validate *iBranchy* for different edge-based requirements. The raw pcap files are processed to create flows for encoding with network-independent header fields and payload data similar to *iknight*. We implement our model using PyTorch [84] and also using code repositories [85, 76]. We use an NVIDIA-based GPU Server with 32 GB RAM to perform our experiments, we also use 80% of data for training and 20% for testing our model. For evaluating *iBranchy*, we consider multiple performance metrics and flexibility-based scenarios. First, we assess the ability of *iBranchy* to exit in earlier exits for the majority of the testing data. Next, we measure *iBranchy* in terms of resource utilization and power consumption and finally assess its flexibility to provide a trade-off between accuracy and edge resources based on the context. We discuss each of these evaluations below.

5.5.1 Significance of Early Exit on IoT devices Network Traffic

Figure 17 shows the percentage of exits from the different exits of *iBranchy* during the training. The number of exits from Exit #1 increases as the training progress, while the number of exits from Exit #3 or the final exit of the network decreases. Therefore, *iBranchy*, over time, learns to extract the significant features for device type inference in the early layers of the DNN. Figure 18 shows the percentage of exits for each device type. The number of exits for device types from Exit #2 is not that significant compared to the number of exists from Exit #1. The less significant exists from Exit #2 could be either because of the choice of the network design or the second branch has not captured enough features for the remaining of the device type samples. *Therefore, the distribution of device exits across different branches is determined by the network design and features of the IoT network dataset.*

5.5.2 Performance of ***iBranchy***'s Edge Deployment

Table 6 shows the runtime performance of *iBranchy*. The results showcases that it performs significantly better with edge-based requirements compared to our baseline DNN. The power consumption of the *iBranchy* model layers decreases by 35.84% than the baseline model layers. Therefore, *iBranchy* uses fewer network components for computing the inference when compared to the baseline IoT DNN model. Moreover, the run time for *iBranchy* model layers is faster than the baseline model layers by about 34.79%. Furthermore, the accuracy drop for the *iBranchy* model is not that significant, as it only drops by about 2%. *Therefore, iBranchy achieves efficient accuracy with*

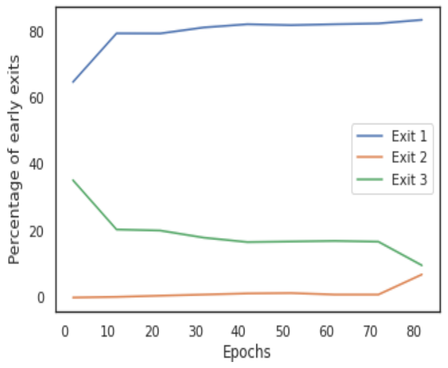


Figure 17: The change in *iBranchy* exits through training

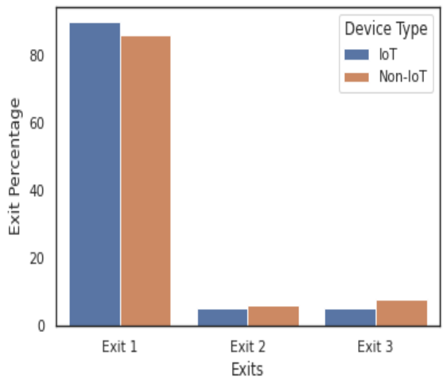


Figure 18: The percentage of exits of *iBranchy* for different device types

fewer resources and is more efficient for edge-based IoT network classification.

Model	Accuracy	Power (%)	Inference Time (s)
Baseline DNN	0.9303	100	0.0569
iBranchy	0.9146	64.16	0.0371

Table 6.: Performance of edge deployment with 11000 samples

5.5.3 Flexibility and Adaptability of *iBranchy* to Hardware and Network Conditions

Given that BranchyNet can be configured dynamically to increase or decrease the entropy threshold values during runtime [76], a similar threshold runtime setting can be used to dynamically configure *iBranchy* for various efficiency schemes. For example, *iBranchy* can change to a low-efficiency scheme with low power consumption during low battery status and switch back to a high-efficiency scheme when the battery is high later. Similarly, during low bandwidth due to bad Wi-Fi, the *iBranchy* can change to a low-efficiency scheme with low inference time at the cost of accuracy and switch back to high efficiency with good Wi-Fi. *Therefore, iBranchy is flexible to adapt to different edge resource contexts and choose the appropriate efficiency scheme.* . However, testing different configurations of the thresholds of *iBranchy* for efficiency and performance trade-offs is out of the scope of this work.

5.6 Discussion

In this chapter, we design and implement an D^2NN based Early-Exit classifier *iBranchy* that can do accelerated inference for edge-based IoT and non-IoT device types with fewer resources. Our framework is flexible enough to support a context-driven network traffic classification system based on the edge environment state such as battery status or network conditions given the conditional computation capabilities of the D^2NN models. Moreover, since we implement *iBranchy* using network encoding component that uses raw network bytes of the device network traffic as input, *iBranchy* can support various device categories from multiple vendors without requiring manual feature engineering. Furthermore, given our encoding process considers network header independent fields, *iBranchy* can seamlessly integrate with different network environments. We believe that our results give some early insights on applying D^2NN such as *iBranchy* for edge-based IoT classifiers. For example, the significant number of IoT devices flowing from Exit #1 shows that the later layers can be cumbersome, requiring a better network design. In our future work, we plan to extend our work to more significant device types with more complex features such as devices from the same vendors and analyze the distribution of these device types over the different exits.

Given *iBranchy* performs conditional computation, it can save power and latency. However, given the complete model still needs to be loaded into the memory, a further technique to optimize memory footprint needs to be designed. One of the approaches we plan is to perform knowledge distillation similar to our earlier discussion in *iknight* can be used build a lighter version of the base model of the *iBranchy*. For example, we can first apply knowledge distillation to the base model of the iKnight to get lightweight models for memory, power, and latency, and then the student model can be further optimized by converting into an iBranchy model.

CHAPTER 6

DDAS - DYNAMIC DEEP NEURAL NETWORK ADVERSARIAL ATTACKS FOR EDGE-BASED IOT DEVICES

6.1 Introduction

Recently, adversarial attacks on machine learning DNN models have increased significantly [86], and have become a significant challenge given their realistic and dangerous attack scenarios such as the ability to alter medical diagnosis [87], deceive surveillance cameras [88], and evade intrusion detection systems [72, 73, 74]. In an adversarial attack, the attacker performs perturbations to the input or environmental conditions in order to target the model's accuracy. Given that D^2NN models are expected to be heavily exploited and utilized in many edge-based scenarios such as minimizing communication between the edge and the cloud [89], real-time video-based classification for faster object detection [90], preserving user privacy [54], and so on, understanding the vulnerabilities of these dynamic models to adversarial attacks is very critical. Furthermore, techniques for improving the robustness of the D^2NN models in order to counter these adversarial attacks must also be explored. One of the mitigation approaches could be to use the generated adversarial attack samples to train the D^2NN models to detect them as attack data, an approach similar to that employed for conventional DNN models [91].

Motivated by the various targets of system attacks and in contract to existing state-of-the-art adversarial attacks that are solely focused on model accuracy, we present a new type of adversarial attack, the Dynamic DNN Adversarial Attacks (DDAS), for edge-based D^2NN models that aim to drain the battery, exhaust power consumption, and increase the latency of the IoT/edge devices. We are interested in adversarial attacks that are designed to defeat D^2NN models' objectives of early inference or conditional computation. In BranchyNet, for example, an adversarial attack might be designed to force the model to use most (if not all) of its layers in inference rather than the few early layers and to terminate early. Similarly, for D^2NN models that adopt skipping parts of input space, adversarial attacks could be designed to cause the network to scan large portions or the whole input space.

In this work we develop DDAS attacks and mitigation techniques for D^2NN models. More specifically, we design and develop ***DDAS-EarlyExit*** attack for D^2NN early-exit based models such as BranchyNet using a GAN-based approach. We evaluate DDAS-EarlyExit using various attack metrics such as host device performance (inference time, power consumption), attack quality, and model classification accuracy using our implementation of BranchyNet model; *iBranchy*, under different attack scenarios. Moreover, utilizing adversarial samples generated by our attack model trained on various configurations, we implement and evaluate a robust incremental training strategy for creating resilient D^2NN. We summarize the contributions as follows:

- We present a novel adversarial attack; the Dynamic DNN Adversarial Attacks (DDAS), for edge-based D^2NN models that significantly increase the power consumption of the hosting device and the inference time of the running application.

- We design and develop DDAS-EarlyExit attack for D^2NN early-exit based models using a GAN-based approach as the first step toward realizing the various types of DDAS attacks.

- We evaluate DDAS-EarlyExit using our implementation of the early exit BranchyNet model; *iBranchy*, under different attack scenarios using multiple attack metrics such as efficiency, performance, and quality.

- We implement an effective incremental adversarial robustness training scheme to develop resilient *iBranchy* models against DDAS-EarlyExit attacks.

6.2 Related work

Adversarial attacks on edge-based deep neural networks are currently emerging and are in the infancy stage. In recent work [92], authors classify the different types of possible edge-based adversarial attacks based on the attacker objectives, model access, attack targets, and defenses. The attacker's goals include disrupting functionality, inferring user, and model privacy. Moreover, a summary of multiple recent edge-based adversarial attack-based works has been described including API attacks[93], side-channel attacks[94], and probing-based attacks [95]. Unlike the existing adversarial attack works, we design, develop, and evaluate adversarial attacks that impact the computation, inference time, and power consumption of the D^2NN deployed hosting device/application in this work. Recently, authors in [96] show adversarial attacks can impact the computation of Reinforcement Learning (RL)-based and Markov decision models. However, our work differs substantially from RL-based adversarial attacks in that we focus on attacking the different activations/components of the D^2NN and not on perturbing the environment/policies used by an RL trained agent.

In some recent works, the design of popular conventional DNN models can be inferred using side-channel attacks by measuring the time [97] and power consumption [98, 99]. We believe that these works give some early insights into the possible valuable information for fingerprinting conventional DNN models, which can help in reverse-engineering the DNN network parameters. However, unlike our work, the authors do not perform any adversarial attacks on the accelerated/edge D^2NN models for impacting resource consumption or network latency. However, we note that their work can further expand the possibilities of building different types of Black-Box attacks for D^2NN models where the confidence measures of the inference are unavailable. Very very recently, some researchers have proposed adversarial attacks on Multi-Exit models [100]. However, unlike their work, we use a GAN-based technique for generating adversarial samples in our approach, given GANs have shown good performance with the generation of adversarial samples for traditional adversarial attacks.

6.3 Background

6.3.1 Adversarial Attacks on Conventional DNN Models

In recent years, significant progress has been made on adversarial attacks [101] aiming at disrupting conventional DNN tasks such as regression, classification, etc. [102] by perturbing the input space, resulting in incorrect inference or output by the DNN network. Adversarial attacks are classified into three main categories [102]: poisoning attacks that target the model during the training process [103], exploratory attacks that exposes knowledge about the underlying model [104, 105], and evasion attacks that targets mis-classification during the testing phase [106]. The proposed DDAS-Early-Exit attack comes under the category of evasion attacks since it aims at perturbing the input samples in order to increase resource consumption and/or latency, as discussed later in section 6.5. In general, perturbations or noise applied to the input space in adversarial attacks should be in a minimal magnitude in order to retain the structure or quality of the input. The effectiveness of evasion attacks varies depending on the available access to the target DNN model and is classified into two types: White-box and Black-box attacks. In White-box attacks, attackers are fully aware of the underlying model architecture, parameters, training data, and the loss function. In contrast, in a Black-box attack, the attacker only has access to the model parameters or confidence scores. We utilize Generative Adversarial Networks (GANs) in developing the proposed attack because of its effective success rates [107, 108].

The adversarial learning approach of GANs has shown to be effective for implementing adversarial attacks against popular conventional DNN models[107]. Inspired by the efficiency of the GAN-based adversarial attacks on popular DNN models, we adopt a similar technique used in AdVGAN [107], a recent GAN model for adversarial attacks, and extend it with our design and implementation of new loss functions that are explicitly designed for the GAN-based DDAS attacks. In the next section, we discuss the threat model of DDAS attacks. We also build a *iBranchy*, a BranchyNet based implementation of a popular image classification DNN architecture RESNET [56] as our target model for GAN-based DDAS-EarlyExit attacks. We discuss the implementation of *iBranchy* in much detail later in section 6.5. It is to be noted that this *iBranchy* is designed and build on a different architecture and dataset than the one we discussed earlier chapter 5.

6.4 DDAS Attack Threat Model

Unlike the traditional adversarial attacks, the attacker objective in DDAS attacks is to compromise the flexibility and dynamic aspects of the D^2NN models by causing the models to activate and exhaust the largest possible portion of the model during the inference process. Consequently, in addition to impacting the model accuracy, DDAS attacks result in increasing the average inference time and computing power consumption of the target D^2NN model. With the significant growth of various edge-based IoT applications and their increasing adoption of D^2NN models [89, 90], DDAS attacks could jeopardize the time-sensitive functionality of IoT Applications/devices causing catastrophic threats. In this study, we focus on BranchyNET as one of the popular D^2NN models. DDAS attacks on *iBranchy*; our realization of BranchyNET, would translate into targeting the model capability to perform early inference at earlier layers and forcing the model to take the longest possible execution path of inference computation by bypassing most or all of the exit points till the inference reaches the natural end point of the model. Henceforth, we refer to attacks that target this early exit capability in particular as DDAS-EarlyExit attacks.

In DDAS-EarlyExit attacks, the core objective of an attacker is to impact the early exit inference capability of the targeted model. We assume that the attacker has access to the one of the data feeds (e.g. a camera) used for inference by the target model and is capable of modifying the input samples by adding adversarial noise. This can be used to impact the Quality of Service (QoS) in multiple ways. A DDAS-EarlyExit malware for a time-sensitive application can cause a severe disruption at critical juncture. Fully autonomous vehicles are an example of this, as they rely on fast input processing to execute critical operations such as obstacle avoidance and traffic mapping. A decreased responsiveness on these functionalities can cause traffic accidents. A recent incident related to Tesla full self-driving computer installations shown that a delay in traffic mapping and pedestrian recognition can potentially lead to disastrous consequences [109]. Another DDAS-EarlyExit attack can target long term power draining attacks on IoT devices deployed with limited energy capacity by continuously exposing them to adversarial samples that increases average power consumption, causing service outages to occur sooner than planned. This can be applied to agricultural, industrial or field-research applications. Because IoT devices used in these applications are more power restricted, exceedingly high power consumption might decrease the life duration of IoT devices, resulting in service disruptions.To be able to perform attacks impacting QoS in such divergent ways, DDAS-EarlyExit design must be flexible in its capability to attack the target model and be able to adapt to different attack scenarios.

Similar to traditional adversarial attacks, DDAS-EarlyExit attacks emphasize the importance of maintaining the visual structure and similarity as well as attempting to maintain classification performance similar to the original clean dataset in order to decrease the chance of detection. Therefore, the perturbations added to the input samples need to be imperceptible while also attempting to nudge the model towards delaying the inference of the samples without impacting the classification output of the model.

In our design of DDAS-EarlyExit, we assume that the attacker has access to the logits output of the different *iBranchy* exit points and would also need the *iBranchy* target model during attack model traiining and therefore we consider our DDAS attack to be a *semi-whitebox* attack. The number of branches used by *iBranchy* and certain information about the configuration used by the target model (e.g. the relative positions of different exit points) can be identified through side-channel inferences or by clustering the inference times[97]. For this work, we also consider that the attacker has access to the target model training data given that such information can be leaked through side-channel attacks. However, even without access to the training data, the attacker can still build an effective DDAS attack by capturing the testing samples and their corresponding outputs over time in a production environment.

6.5 DDAS-EarlyExit Attack Design

The attacker's main objective in the DDAS-EarlyExit attack is to increase the entropy of the perturbed input sample leading to a delayed inference of the sample in an early-exit model. However, depending on the attack scenario discussed in section 6.4 the attacker may choose to trade-off between attack performance and accuracy and therefore might need more control on the impact of entropy at different exits. Moreover, the adversarial sample should have minimal noise/perturbation to preserve the structure or quality of the input. Furthermore, the DDAS-EarlyExit attack should generate the perturbation based on the type of the input sample and should understand the mapping between the structure of the data and corresponding perturbations and apply the same during inference phase without any access to the target early-exit models. Therefore, we design the DDAS-EarlyExit attack model with the following objectives 1) Fine grain control of the impact on entropy based on attacker objectives, 2) minimize the perturbation, and 3) preserve the functionality/structure of the original input. Given these objectives, we choose to implement our attack model training as an adversarial learning process using GANs [18], given their ability to generate high-quality synthetic data and preserve the original data's properties. In this section, we first design DDAS-EarlyExit loss function that will optimize attack model training based on attacker objectives and then we discuss the implementation of the DDAS-EarlyExit attack model.

6.5.1 DDAS-EarlyExit Loss Function

The standard GAN Network based adversarial training consists of a min-max problem to minimize the generator loss and maximize the discriminator loss during training. A small loss value for the generator indicates high-quality synthetic samples and whereas a high loss value for the discriminator indicates a high misclassification rate of synthetic samples with the GAN network training converging when both the losses converge. In a recent extension of the GAN for adversarial attacks on conventional DNNs, the authors extend the GAN architecture with the additional component target DNN and an adversarial loss to measure the ability of the generator to produce high-quality perturbations [107]. The choice of the adversarial loss can be different based on the optimization problem. For example, the popular loss function $C\&W$[110] can be chosen for a non-targeted adversarial attack that maximizes the probability of any other class label prediction. Similarly, the target model loss function may be efficient to perform targeted adversarial attacks that minimize the distance between the predicted class and targeted attack class[107]. However, both $C\&W$ loss and targeted loss functions will increase the confidence of the false label and lead to an early exit. Therefore, unlike traditional miss-classification losses that decrease the entropy, in our implementation of the DDAS-EarlyExit loss, we need to use an entropy-based loss that can maximize the entropy at exists for the perturbed inputs produced by the generator.

We choose to implement a DDAS-EarlyExit attack using a recent GAN-based adversarial attack model approach [107] where the model is able to achieve a high attack rate against traditional DNN models but with minimal perturbation. Their attack model is trained to minimize the GAN network loss, which is a combined loss for the discriminator loss, generator loss, perturbation loss used to limit perturbation range and misclassification loss. Since early-exit model such as BranchyNet uses entropy-based metric to make a decision whether to perform an early exit inference, the DDAS-EarlyExit attack should generate perturbations that aims to to have high entropy at the early exit points and ultimately causes the inference to happen at later exits, therefore in our attack model loss we need to use entropy based loss instead of misclassification loss. Moreover, since our objective is to train the attack model based on attacker objectives and therefore we design our DDAS-EarlyExit attack loss with a configurable approach that can tune the attack model to have different entropy impacts at different exits and hence divide the our attack model loss into two parts. The first part of the loss is summation of the perturbation loss, discriminator loss, and generator loss similar to [107] and for the other significant part instead of the misclassification loss we use configurable entropy loss (CEL) defined in Equation 6.1 which is a combination of weights of the entropies of different exits across the early-exit model where weights represent the magnitude to either maximize or minimize the entropy at an exit.

$$Loss(x) = \sum_{i=0}^{n} W_i * Entropy_i \tag{6.1}$$

This fine-grained exit-specific CEL entropy loss has multiple benefits in controlling the accuracy and attack success. For example, if the attacker chooses to have a very high attack rate, the CEL loss can be configured to maximize the entropy over all the exits. However, in stealth mode, the attacker may choose to have high accuracy with a low attack rate and therefore may only need CEL entropy loss with respect to the first exit of the network. Similarly, the attacker can also choose a maximum possible accuracy and maximum attack success by taking a configuration of CEL loss with maximizing the entropy across all exits but minimizing the entropy at the last exit as all the samples will be pushed to the last exit with less drop in overall accuracy. Moreover, the attacker can have some weights for the entropy of each exit and an in-depth study of such weight combinations would have to be performed. We do note that our attack could use power or inference-based loss instead of entropy loss, CEL, in an attack scenario where access to the entropy values are unavailable. However, such an attack design is challenging with GANs and is out of the scope of this work. Given our DDAS-EarlyExit loss function can handle different attacker objectives we discuss the training and implementation of our GAN attack model using this loss function.

6.5.2 DDAS-EarlyExit Model Implementation

Figure 19 shows our attack model implementation where we use a generator and discriminator architecture similar to [107] but a different type of target model - *iBranchy*. We implement a BranchyNet based target model *iBranchy* using the ResNet32 classifier [56] trained on the CIFAR-10 dataset with 50000 training samples[58] and configure it with three exits: Exit #1, Exit #2, and Exit #3, at different layers of the network. Exit #1 is configured at the end of the first residual block, Exit #2 at the end of the second residual block, and Exit #3 at the end of the network. We chose a lightweight design for both the branches configured at Exit #1 and Exit #2 with a pooling layer followed by a flattening fully connected layer to add less overhead at run time to simulate edge deployments. In our attack model training approach, we make some significant changes compared to [107]: first, as we use an early-exit model as we depend on the inferences phase outputs of the different exits and also unlike their attack objective which is to maximize the confidence for the non-target class using a misclassification loss, in our training we utilize a configurable entropy loss as shown in Equation 6.1 to control/maximize/minimize the entropy based loss for the generator across different exits and the DDAS-EarlyExit loss for overall GAN network training. We use CIFAR-10 training dataset with 50000 training samples, same as the *iBranchy* model training, to train the generator for generating noise to perturb the the clean images which are are then discriminated by the discriminator to differentiate between perturbed and clean images and also the entropy across different exits of *iBranchy* are calculated. It is to be

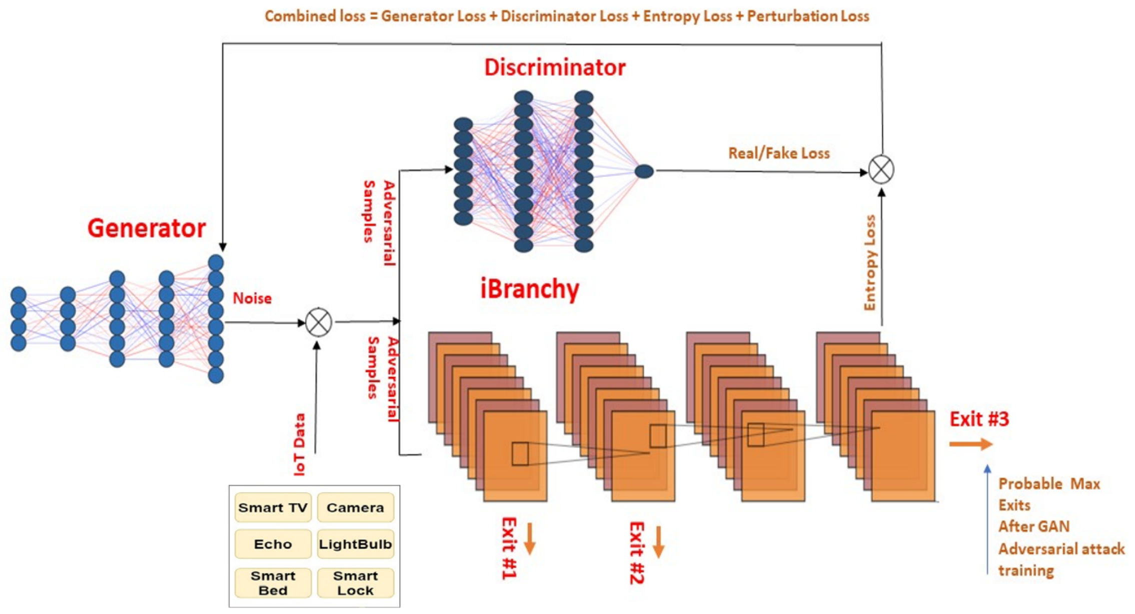


Figure 19: DDAS-EarlyExit-GAN Attack Model

noted access to entropy across all exits for a sample are only needed at attack model training and during the inference phase the attack model is completely independent and can generate perturbations without any access to the exits or the *iBranchy* target model.

The approach we have taken for mitigating the DDAS-EarlyExit attack for D^2NN is adversarial retraining similar to [91]. Adversarial retraining generally works by generating adversarial examples and then mix them with clean examples and use both adversarial and clean data to retrain the target model and has shown to build effective robust models. However, the goal of our DDAS-EarlyExit is different than misclassification and its efficacy is measured through different metrics. Therefore, we aimed to implement iterative robustness retraining scheme and evaluate its effectiveness on our DDAS-EarlyExit.

The iterative robustness process we implemented in Algorithm 2 works as follows. Step 3: we train DDAS-EarlyExit to target the classification model M_i by providing it with training dataset samples and train the generator GAN to generate the correct attack noise for DDAS-EarlyExit attack. The attack noise is then added to the clean image, becoming the adversarial image, and then the adversarial image passes by the

GAN discriminator to check whether it can differentiated from clean samples. This operation continues over epochs until we obtain an adequate attack model A_i, we then proceed to the next step. In Step 4: we test the robustness of the classification model M_i against the samples generated from GAN attack model A_i by calculating both accuracy and samples distribution over all exit points. If the metrics shows adequate robustness that exceed a predefined threshold T we stop the iterative robustness process, otherwise we proceed to the next step. Step 5: we perform adversarial training on the classification model M_i to boost its robustness by doing the following: We take the clean images from the training set and then create their adversarial version by adding GAN-generated noise for each of these training set images. More specifically, while training the model with a batch of samples (128 images/batch), we take the 128 clean images and create 128 adversarial versions using GAN noise specific to these images. Now we finally pick 128 images randomly out of the 256 mixed images(clean and adversarial) and train the model. This training continues for all batches of an epoch (total images of the training set split across batches) and is again repeated for different epochs till we obtain a satisfactory model M_{i+1}. The output model M_{i+1} from this training process is then validated for robustness against the attack model A_i to ensure that robustness goals have been achieved. The results of this approach and its efficacy will be discussed in the evaluation section 6.6.

6.6 Evaluation

In this section, we discuss the experiments to evaluate our DDAS-EarlyExit attack and iterative robustness defence approach for *iBranchy* models. We implement the DDAS-EarlyExit attack using PyTorch [84] and building on top of code repositories for BranchyNET implementations [76] [85] and AdvGan [107]. We design our experiments with different DDAS-EarlyExit loss configurations and evaluate the impact of attack performance for different metrics such as accuracy, samples exit distribution, power

Algorithm 2 Implementation of *EarlyExit-attack-robustness* adversarial training

```
1: procedure ADVERSERIALROBSUTENSS
2:     while $i \neq$N do
3:         Use DDAS-EarlyExit to generate an attack model $A_i$ targeting $M_i$
4:         Test the robustness of the model $M_i$ against $A_i$
5:         if $M_i$ passes robustness threshold $T$ then break
6:         end if
7:         Use $A_i$ to generate adversarial samples $S_i'$ and retrain to obtain a more robust $M_{i+1}$
8:         Validate the new model $M_{i+1}$ using attack model $A_i$
9:     end while
10: end procedure
```

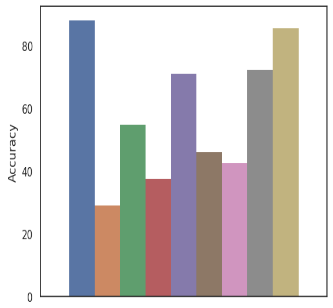


Figure 20: The change in *iBranchy* accuracy under DDAS attacks.

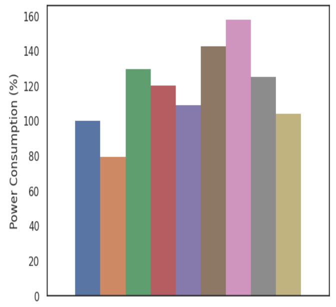


Figure 21: The change in *iBranchy* power consumption under DDAS attacks

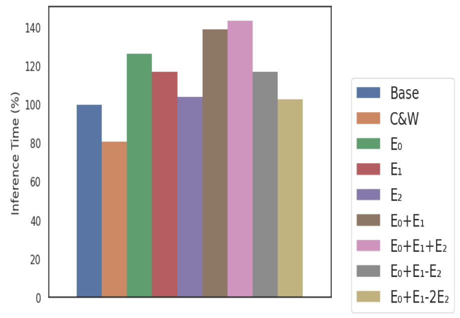


Figure 22: The change in *iBranchy* inference time under DDAS attacks

consumption, and inference time. We then perform iterative robustness training on *iBranchy* with the help of DDAS-EarlyExit to evaluate the effectiveness of adversarial robust training. We run our experiments on both GPU-enabled local machines and Amazon-EC2 instances.

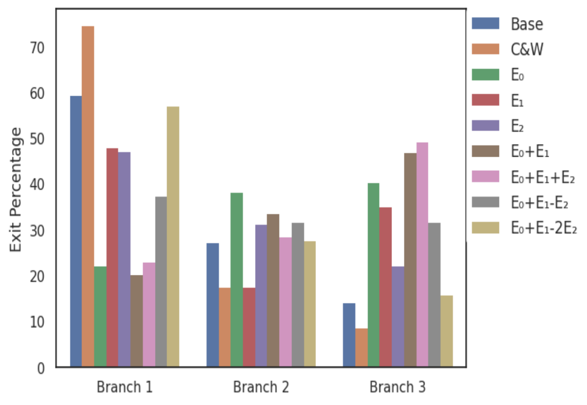


Figure 23: The change in *iBranchy* branch level exit distribution different DDAS attacks

6.6.1 Attack Performance on D^2NN early exit models

We evaluate the performance of DDAS-EarlyExit using four different metrics: overall accuracy of the model after the attack shown in Figure 20, average power consumption of the model shown in Figure 21, average inference time of the model shown in Figure 22, and output exit distribution over the different branches shown in Figure 23. Using these metrics, we declare that an ideal attack should be able to maximize power consumption and inference time at a minimal or no cost to accuracy. The evaluation is performed by measuring these metrics over 10000 test samples.

We evaluate the DDAS-EarlyExit attacks with the following adversarial entropy maximizing loss functions $entropy\_E_1$, $entropy\_E_2$, $entropy\_E_3$, $entropy\_(E_1 + E_2)$, $entropy\_(E_1 + E_2 + E_3)$, $entropy\_(E_1 + E_2 - E_3)$, $entropy\_(E_1 + E_2 - 2 * E_3)$. The notation used identifies which exit we are targeting for entropy maximization. For example, the configuration $entropy\_(E_1 + E_2 - E_3)$ will aim to maximize the entropy at Exits #1 and #2 and minimize it at Exit #3. For the sake of completion and to provide a base model for comparison we also include a classic misclassification adversarial attack using $C\&W$ loss[110] to highlight the difference between standard adversarial attacks and our DDAS-EarlyExit attack.

First, starting with the standard adversarial attack using $C\&W$, this attack aims to maximize the misclassification loss at Exit #1. The results show a high negative impact on the accuracy of the model dropping it from 89% to about 28% as seen in Figure 20, while at the same time causing more samples to leave from the earlier branch to increase; increasing the percentage of samples leaving from the first branch from 59% to 74%, and decreasing the percentage of samples leaving from the last branch from 14% to 8%. This in turn causes a *decrease* in both average inference time and power consumption by about 21% and 20% as seen in Figures 22 & 21 respectively. This shows that standard adversarial attacks are not capable of achieving the goals of our attack of negatively impacting power consumption and inference time while maintaining high accuracy, on the contrary it does the opposite, it decreases accuracy and power consumption and inference time. We attribute this ineffectiveness to the maximizing misclassification confidence does not contribute to entropy maximization and can sometimes even lead to minimizing it.

Now we move on to analyze the different entropy based adversarial loss function outlined before. Starting with $entropy\_E_1$, we immediately notice the difference between this and the previous $C\&W$ based attack. As this one causes a massive decrease in the percentage of samples leaving from the first branch from 59% to 21%, second branch increase from 27% to 37% and last branch increase from 13% to 40%. This in turn causes an increase in the inference time and power consumption by 26% and 30% respectively. The overall accuracy drops but by a much lesser degree to 54% up from 29% in the $C\&W$ based attack. Changing the function to $entropy\_E_2$, we notice a decrease in exit percentage of the second branch from 27% to 15% , first branch from 59% to 47% and last branch increase from 13% to 34%. The drops in inference time and power consumption are less severe than the previous attack due to the fact that not many samples exiting for the first branch are impacted by this one, both were valued at 16% and 20% respectively. Moving to $entropy\_E_3$, we notice that this attack is less significant than the previous ones as maximizing the entropy of the samples at the last exit has smaller value than earlier branches due to the simple fact that the samples have no choice but leave at this point anyway making the impact of this particular attack weak comparatively. *Therefore, we conclude that maximizing the individual exit entropy at later exits will reduce impact of DDAS-EarlyExit attack.*

Moving on to the next adversarial attack *entropy_*$(E_1 + E_2)$, this proves to be more successful than our previous attempts, the distribution of the exit samples across branches changes from 59%, 27%, 14% to 19%, 34%, 47%. Demonstrating a more significant increase in the load at the later exits. Due to the summation of the entropies, the attack is also more potent in impacting our performance metrics. The trend continues with our most aggressive attack *entropy_*$(E_1 + E_2 + E_3)$, which adds the entropy of the last exit as well. This attack have an even stepper effect on the sample distribution with more samples leaving from the last branch. We conclude from this that combining the entropies proves to be an effective way to increase the potency of the attack. However, this comes at the cost of higher drops in the accuracy. Our next two attacks aim at finding a balance between attack aggressiveness and accuracy drop. The first attack *entropy_*$(E_1 + E_2 - E_3)$ aims at maximizing the entropies of the first and second branches while decreasing the entropy of the third one. The reasoning behind this approach is that while we aim to increase the entropies of the first two exits to nudge the model towards moving samples into later exits for evaluation, we also want to keep inference accuracy high and we theorize that decreasing the entropy output of the last layer may lead to the model to be more accurate. The results of this approach are a less stepper drop in accuracy to just 72% at the cost of much less aggressive attack as just 32% exit from the last branch. This shows the flexibility of this attack, as this can be considered as a hyper parameter that can be used to tune the aggressiveness of the attack. Finally to take this further, we test *entropy_*$(E_1 + E_2 - 2 * E_3)$, which aims to further minimize the classification accuracy drop. The results here are the closest to the base model with minor changes across the board. We believe that our results shows the both the potency and flexibility of our attack and more importantly highlights the

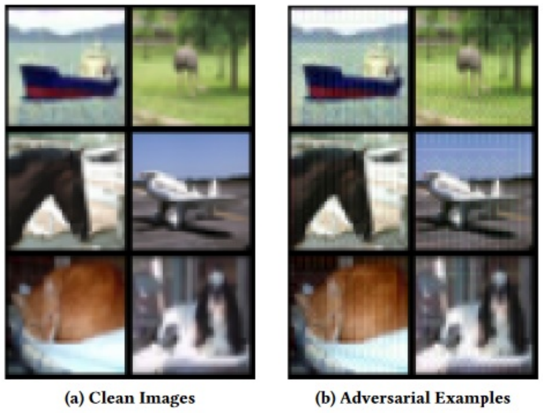


Figure 24: An illustration of our DDAS-EarlyExit attack

role of an efficient loss configuration and how it can support a controlled attack rate scenario. Moreover the impact of this attack visually is negligible compared to cleans samples as shown in Figure 24

6.6.2 Evaluation of Iterative Robustness Training

For building defences to the early attacks we implement an iterative robust adversarial training . Table.7 shows the effect of the adversarial robustness retraining on the model. As we can see the first iteration of the iterative robustness performs surprisingly well against adversarial attacks from the same GAN model used during its robustness retraining. It shows the model restored to the base model basically when it comes to performance metrics as it shows very similar branch exit distribution, accuracy, inference time and power consumption but with the added robustness against DDAS-EarlyExit attacks. *Therefore adversarial samples generated using DDAS-EarlyExit-GAN is effective to enhance the robustness of the early-exit models.*

However, the effects of robustness degrade when the attackers can further train DDAS-EarlyExit attack aiming to target the robust model. But we also note that, in the second iteration, the drops in performance metrics are significantly smaller than in the first iteration as accuracy drops to only 62% instead of 39% and smaller impacts on branch exit distribution, inference time and power consumption. This shows that the robust model still retains some of the capabilities against further attacks. *Therefore adversarial robustness of the early-exit models trained using DDAS-EarlyExit-GAN will be effective for unknown DDAS-EarlyExit attacks.*

Model	**Acc**	**Exit #1**	**Exit #2**	**Exit #3**
Base Model	88%	59%	27%	13%
Attack iter 0	39%	26%	24%	50%
Robustness 0	85%	56%	26%	18%
Attack iter 1	62%	30%	26%	43%
Robustness 1	84%	56%	28%	15%
Attack iter 2	68%	35%	28%	37%
Robstness 2	83%	54%	25%	21%
Attack iter 3	72%	40%	27%	33%
Robustness 3	83%	55%	26%	19%
Attack iter 3	73%	41%	30%	29%

Table 7.: Performance for edge deployment across 20000 samples

6.7 Conclusion and Future Work

We design and develop DDAS-EarlyExit attack, a GAN based attack against dynamic edge deployed deep learning networks utilizing early exit models such as *iBranchy*. We show that our attack is capable of significantly impact the performance of these D^2NN models in terms of power consumption and inference time latency for edge-based IoT application/devices. We discuss different attack scenarios utilizing DDAS-EarlyExit and show how these scenarios can play out causing catastrophic disruption in the QoS. We also evaluate different entropy based adversarial loss functions and highlight how they differ from misclassification loss, and we showcase how they can be used to allow for a flexible attack and how we can fine tune its aggressiveness to either hit its performance metrics or choose to relax its goals for a more stealthy approach in the hopes of evading detection. These flexibility is gained by our design of the adversarial loss function. Finally, we implement and evaluate an incremental adversarial robustness retraining scheme for building resilient early-exit type D^2NN models. We showcase the effectiveness of our defense and we also highlight its limitations. Furthermore, we also plan to design and develop DDAS attacks for other D^2NN models and evaluate their defenses. Finally, we believe this work will trigger discussion for adversarial attacks and defenses on D^2NN models and help with the new research directions. Next, in chapter 7 and 8, we build the components of the *iPrivacy* service *PrivacyGuard* and *MirageNet* to counter privacy based attacks.

CHAPTER 7

PRIVACYGUARD: EXTREME SDN FRAMEWORK FOR IOT AND MOBILE APPLICATIONS FLEXIBLE PRIVACY AT THE EDGE

7.1 Introduction

In addressing the security concern with side-channel analysis, several techniques like perfect secrecy theory [111], mix based systems and anonymous systems [112] are proposed to hide traffic signatures and characteristics in order to make them less identifiable. The most popular techniques are based on traffic shaping like traffic padding [113, 114, 115, 116, 117], faking superfluous packet and chopping packets into fixed size segments [115], and traffic morphing [115, 118]. The performance of these traffic shaping techniques in terms of efficiency and overhead varies based on their configuration parameters. For example, the efficiency of the traffic padding approach in obfuscating the traffic signature, as shown later in experiments, increases with the percentage of the padded traffic packets. However, this higher efficiency comes with higher overhead in terms of network bandwidth and power consumption since more bits are transmitted. For example, while a padding configuration providing high obfuscation efficiency with high overhead is suitable for networks with low traffic loads, it significantly degrades the performance of a highly saturated networks and, consequently, switching to a lower overhead configuration would be more desirable.

Motivated by the above observations, in this chapter, we design, develop and evaluate a flexible and programmable privacy preserving framework, *PrivacyGuard* that is inspired by our another vision of pushing the Software Defined Network (SDN)-like paradigm all the way to the wireless network edge [119, 120]. This vision is realized by extending and deploying SDN components (e.g., Open vSwitch [121]) on mobile devices and WiFi Access Points (in home or campus environments) or Proxy server (in open and public WiFi Hotspots). We refer to these SDN components on mobile end devices and access points as *extreme SDN* to differentiate them from the traditional SDN used in the network core. In our approach, the proposed extreme SDN works independently and without any collaboration or support from the network core SDN. The basic idea of *PrivacyGuard* framework is to create one or more vertical network slicing between mobile devices and WiFi APs/Proxy server corresponding to one or more rules (policies), which applies the optimum per-flow/per-application privacy preserving scheme based on application requirements, user objectives, device characteristics, and network conditions in real-time fashion.

We summarize the contributions of this chapter as follows:

- We design and develop *PrivacyGuard*, a privacy preserving framework to obfuscate the activities of sensitive IoT and mobile applications from adversarial attack over encrypted (or unencrypted) WiFi network. The framework supports important features such as flexibility, transparency, real-time adaptability, and context-awareness.

- We realize and implement a prototype of *PrivacyGuard* on Android mobile devices that enable us to apply per-application *traffic shaping* and IPsec tunneling schemes.

- Finally, we evaluate and analyze the performance of *PrivacyGuard* using different applications on mobile devices to evaluate its efficiency, energy consumption, network overhead.

7.2 Threat Model and Applications/Flows Identification

We consider the privacy threat model where the adversary passively eavesdrops the encrypted/unencrypted wireless traffic between the mobile devices and the WiFi Access Points (APs). In IoT context, the mobile devices act as gateways for the IoT devices that utilizes non- WiFi technologies, such as Bluetooth Low Energy (BLE) or Z-wave, to communicate with their corresponding applications on the mobile devices. By eavesdropping, the adversary can extract and analyze the WiFi side-channel information such as packet sizes and inter-arrival packet times to identify the running mobile applications and the corresponding usage activities [24].

In this work, we refer to these applications as the *sensitive applications* in which the goal of the adversary is to identify the usage of these sensitive applications (e.g., mHealth apps, IoT apps, etc.) to infer user's information. In this work we offline build a C5.0 decision tree and a k-NN (k=3) classifiers based on the statistical features of the side-channel information for identifying the active applications and their corresponding flows in real-time [120]. In this work, we assume the adversary uses these classifiers to identify the active applications/flows. In addition, we use these classifiers in our experiments to evaluate the efficiency of the used obfuscation schemes.

7.3 Related Work

In order to protect the application's network data, there have been many proposed solutions for managing network-wide mobile devices from network infrastructure [122]. However, such remote network management solutions are not well-suited for dynamic network devices like mobile devices. Therefore, researchers are focusing recently on client-side network security solutions [123, 124]. Among these works, very few ones have fine-grained and programmable network security policies targeting the applications. For instance, one of these solutions provides application specific and device-context-aware network access policies [123] . In another work, the authors have used network virtualization technique, similar to *PrivacyGuard*, to isolate the network traffic between sensitive (i.e., medical applications) and non-sensitive applications [124]. However, unlike *PrivacyGuard*, none of the client-side security solutions have focused on the network security concern of the side-channel attack for sensitive mobile applications.

Previously, numerous works have addressed the eavesdropping attack [23, 24] based on side-channel information. However, very few works have actually proposed and validated the use of *traffic shaping* techniques to address eavesdropping attack [115, 114]. However, none of these works has considered IoT devices and mobile device applications. Recently, researchers were able to show how a side channel attack could identify IoT devices, and then proposed a traffic shaping technique based on rate-reshaping [25]. This technique uses Independent Link Padding based approach that transmits fixed size packets at a constant rate. Given the programmability of *PrivacyGuard*, it is straightforward to adopt this technique as well as any other *traffic shaping* technique into *PrivacyGuard*.

In another work, authors in [125] apply traffic demultiplexing at the MAC layer to protect the Wi-Fi traffic, which requires expensive MAC layer management between mobile devices and access points (APs). In addition, this technique requires modifying the wireless device driver for supporting the multiple virtual interfaces and distributing the traffic over these interfaces. *PrivacyGuard* doesn't require any driver level modification and it is not limited to any specific Wi-Fi AP configuration.

7.4 ***PrivacyGuard*** **Objectives**

In considering the above threat model, we design *PrivacyGuard* with the following objectives.

Flexible per-application per-flow privacy preserving schemes: Given different applications have different sensitivities and requirements, *PrivacyGuard* should have the flexibility of applying different privacy preserving schemes to different applications. Moreover, different applications and even different flows of the same application would have different traffic characteristics and, consequently, would require different schemes to obfuscate the application. For example, Dropbox generates two flows for uploading/downloading a file, wherein one direction data packets are at its maximum possible size, while the other direction contains just identical TCP ACK packets. Therefore, the TCP ACK flow should use a scheme that pads these TCP frames to look like the data packets flow, which might not need any padding scheme. Therefore, *PrivacyGuard* should be designed, through introducing new action commands in Open vSwitch (OVS), as we will describe later, to support applying per-application per-flow configurable schemes.

Programmable privacy preserving policies: Given the performance of privacy preserving schemes (e.g., traffic shaping schemes) depends on their configuration, *PrivacyGuard* should support programmable APIs to define and configure different schemes dynamically. In addition, it also should support to define set of rules (policies) that map individual applications/flows to their optimum schemes based on on the application, user, device, and network conditions and characteristics in real-time fashion.

Context aware privacy preserving policies: Different application requirements, user objectives, device characteristics, and network conditions, which we refer to them as *contexts*, require different performance levels of the applied privacy preserving schemes. Therefore, *PrivacyGuard* should support to integrate the context into the defined policies in order to select in real-time the optimum scheme for individual applications/flows that adapt to the given context.

Policies are transparent to applications: Unlike previous systems that often require redesigning both the client side and the server side of the application *PrivacyGuard* should seamlessly support any application without requiring any modification on either client or server-side of the application.

7.5 Privacy Preserving Schemes

PrivacyGuard adopts and utilizes a number of the popular *traffic shaping* schemes, due to its popularity and simplicity, to obfuscate applications traffic signatures from any adversary.

One of the adopted traffic shaping schemes is ***packet-padding*** that applies a padding bytes to a percentage p of the application traffic packets. Although the selection of these padded packets could follow different distributions, we select these packets uniformly in which each packet of the traffic will be padded with a certain probability p. The size of the padding bytes will follow a random distribution such as Poisson distribution or Gaussian distribution. Note that the actual padding size will depend on the configuration parameters of the distribution. For example, if we set the standard deviation σ and the mean μ parameters of the Gaussian distribution to a very small value and a large value respectively, the outcome padding sizes will follow a large uniform padding distribution. To guarantee that none of the padded packets exceed the maximum transmission unit (MTU), we truncate the outcome padding sizes as needed.

Another popular traffic shaping technique is ***packet-delaying*** that shapes the inter-packet transmission times (IPTs). In doing this, we increase the queueing time of each packet, before sending it down to the WiFi driver, with a random delay selected from a uniform distribution. Similar to *packet-padding* scheme, *packet-delaying* scheme could follow any other distribution.

Recent studies show that over half of the connections made by mobile applications are insecure since they don't use any of the network or application level encryption [126]. Consequently, open or unencrypted WiFi hotspot connections expose several data packet fields such as "IP" header in which the adversary could easily identify the applications/flows from these packets even when we are using any of the privacy preserving schemes. In mitigating this vulnerability, *PrivacyGuard* applies IPSec tunneling scheme between the *PrivacyGuard*'s ends as described in Section 7.5.1, on top of any used traffic shaping scheme, to prevent eavesdropping any of the packet fields. The choice of using tunneling is entirely configurable and would be selectively applied for selected applications based on the network condition (i.e., WiFi is either open or unen-

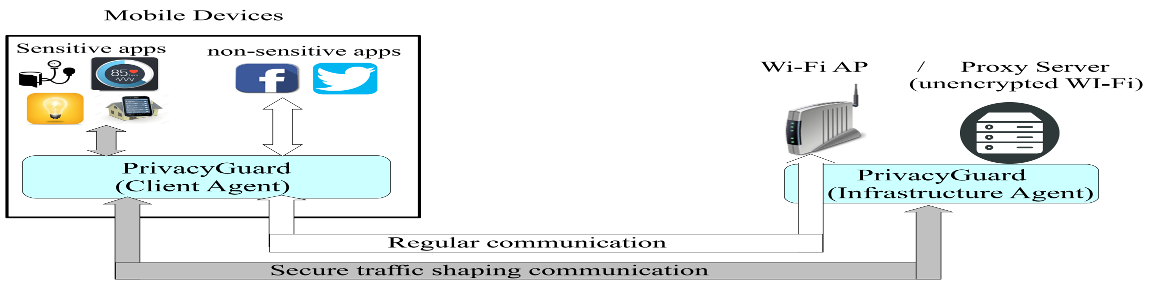

Mobile Devices
Sensitive apps
non-sensitive apps
PrivacyGuard (Client Agent)
Wi-Fi AP / Proxy Server (unencrypted WI-Fi)
PrivacyGuard (Infrastructure Agent)
Regular communication
Secure traffic shaping communication

Figure 25: Use-case scenario of *PrivacyGuard*.

crypted).

Although the current implementation of *PrivacyGuard* adopts the traffic shaping and IPSec tunneling techniques discussed above as a proof of concept of privacy preserving schemes, *PrivacyGuard* is a flexible framework that easily could be extended to support several other privacy preserving schemes such as injecting fake superfluous packets, chopping packets into fixed-size segments, traffic morphing, etc. Moreover, *PrivacyGuard* enables, through defining flow policies as described later, to configure the applied privacy preserving to adapt its performance to the current context.

7.5.1 *PrivacyGuard* Basic Operation

Figure 25 shows a typical use case of *PrivacyGuard*, which consists of two edge agents i) Client Agent, and ii) Infrastructure Agent that run at two different ends as shown. The client agent always runs on the user's mobile devices with system-level permission. However, the place of running the infrastructure agent depends on the configuration ability of the WiFi APs. In environments, where it is a common practice to manage and configure the WiFi APs (i.e., home, campus, office, etc.), infrastructure agent runs on the APs. On the other hand, where it is less common to manage and configure the WiFi APs (i.e., public hotspots, coffee shop, airport, etc.), infrastructure agent runs as a proxy server in the cloud. Note that running these agents on edge devices is similar to the Bring-Your-Own-Device (BYOD) model that is widely widely accepted by the community. Moreover, the community has accepted several works recently to adopt SDN on WiFi APs [127] and mobile devices [119, 53].

In *PrivacyGuard*, both of these agents agree on the *traffic shaping* policies per-application per-flow as well as the symmetric keys for IPsec tunneling between the two agents. While the *client agent* applies *traffic shaping* policies on the uplink network flows generated from the mobile devices, the *infrastructure agent* applies *traffic shaping* policies on the downlink network flows to the mobile devices. Note that, *PrivacyGuard* only applies policies on the selected set of sensitive mobile applications provided by the user. In *PrivacyGuard*, both agents utilize Open vSwitch (OVS) to set and enforce *traffic shaping* policies correctly on the network flows from/to sensitive mobile applications as we describe in the next section. It is important to highlight that *PrivacyGuard* is entirely transparent to applications and networks in which it can be seamlessly deployed into any network configuration without requiring any support from application providers.

7.6 Performance Evaluation

7.6.1 Experiments Setup

In our experiments we use a Nexus 4 smartphone with Android 4.4 running *PrivacyGuard* client agent as a user device, and an Ubuntu 16.04 laptop with Intel Core i5-2520M @2.5GHz CPU running *PrivacyGuard* infrastructure agent as a Wi-Fi AP. We install 8 commercially available IoT device-based applications on the Nexus device which acts as the gateway. These applications span different domains including home appliance, medical and fitness. We use three different traffic shaping schemes based on *packet-padding* and *packet-delaying.* The first is Norm_Pad, which is a *packet-padding* scheme where the padding bytes size follows a Gaussian Distribution and μ and σ parameters are set to 400 and 100 bytes respectively. The second is Norm_Pad_Delay that applies both a *packet-padding* scheme (same as Norm_Pad), and a *packet-delaying* scheme. In order to calculate the delay, we calculate both the average minimum IPT (*min*) and the average maximum IPT (*max*) of all the applications. Then, we use that min-max range to generate uniformly distributed random delays to extend the IPTs of the targeted application/flow. In our implementation, we set *min* and *max* to 0ms and 20ms respectively. The last scheme is Max_Pad_Delay that pads packets with the maximum possible bytes and also use the *packet-delaying* scheme. We run 100 classification experiments for each application for each of the traffic shaping schemes.

We evaluate our experiments with metrics based on both efficiency and overhead. Efficiency is measured using the *accuracy* and *precision* metrics. The accuracy of a traffic shaping scheme of a specific application is the percentage of the true positive classifications (i.e., the number of the target application classification experiments that are correctly identified the target application) to the total application classification experiments. On the other hand, the precision is the percentage of the true positive classifications to the summation of the true positive classifications and the false positive classifications (i.e., the number of the other applications classification experiments that are wrongly identified as the target application). An application with a low precision indicates that the used traffic shaping scheme confuses the adversary in which it makes him falsely identify a large portion of the traffic of the other applications as the target application. Note that the efficiency metrics are calculated based on the classification models discussed in Section 4.1.

The overhead is measured in terms of: i) the network bandwidth overhead measured as the percentage of the additional bytes sent over the network, and ii) the energy overhead measured the percentage of additional power consumption. To measure the actual power consumption, we connect the battery of Nexus 4 to the Monsoon Power Monitor device.

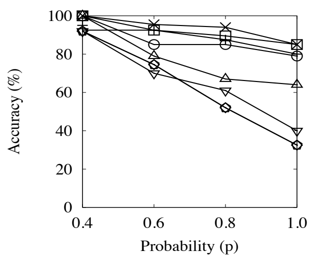


Figure 26: The accuracy of Norm_Pad scheme for different applications and p values.

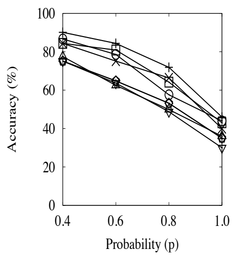


Figure 27: The accuracy of Norm_Pad_Delay scheme for different applications and p values.

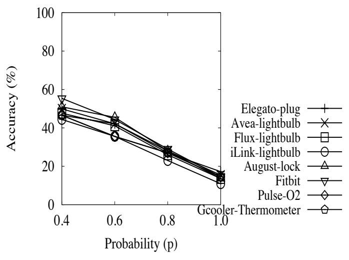


Figure 28: The accuracy of Max_Pad_Delay scheme for different applications and p values.

7.6.2 Traffic Shaping Schemes Performance

In this section, we analyze and evaluate the performance of Norm_Pad, Norm_Pad_Delay, and Max_Pad_Delay traffic shaping schemes as examples of different privacy preserving schemes. Figure 26 shows the accuracy of Norm_Pad scheme with different probabilities (p) for the eight applications. As shown, as p increases, the scheme becomes more efficient in obfuscating the application signature that results in a decreasing identification accuracy. It is interesting to observe that while the scheme has high efficiency for some of the applications such as the *Fitbit* application with large values of p, it fails in obfuscating other applications such as the *Flux-lightbulb* application. We also tried different configurations of the scheme in which it generates similar results.

By analyzing the traffic characteristics of the applications with low efficiency (i.e., Elegato-plug, Avea-lightbulb, Flux-lightbulb, and iLink-lightbulb), we observed that these applications transmit their packets at periodic patterns. Therefore, any privacy scheme that is based only on *packet-padding* will have a low efficiency in obfuscating these applications. However, in Figure 27 that plots the accuracy of Norm_Pad_Delay scheme, all applications show better efficiency (i.e., low accuracy) as p increases. Therefore, *different applications/flows have different traffic characteristics that require different privacy preserving schemes in order to achieve high efficiency.*

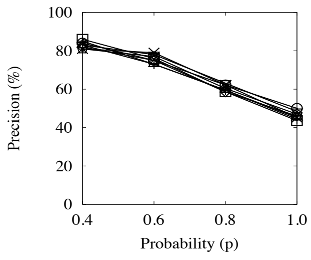


Figure 29: The precision of Max_Pad_Delay scheme for different applications and p values.

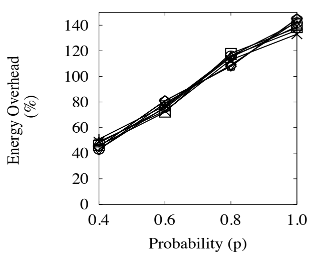


Figure 30: The power consumption overhead of Max_Pad_Delay scheme.

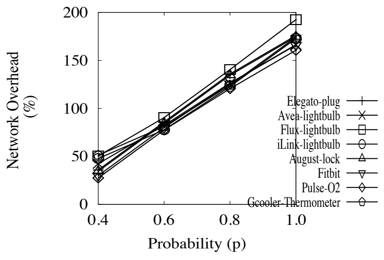


Figure 31: The network bandwidth overhead of Max_Pad_Delay scheme.

Figure 28 shows the accuracy of Max_Pad_Delay scheme in which its efficiency exceeds the other two schemes even at low values of p. This is because some applications such as *Elegato-plug iLink-lightbulb*, and *Flux-lightbulb* transmit many large size packets in which the signature patterns of these traffic are hard to be obfuscated with a padding scheme using few padding bytes. However, when we pad the packets to the maximum possible packet size (i.e., MTU size), it becomes hard to identify the signature of these traffic.

In addition, Figure 29 shows the precision of the same scheme. As shown, the precision drops gracefully as p increases that help significantly in obfuscating the applications/flows by confusing the adversary more. For example, while Figure 28 states that an adversary will be 25% of the time is able to correctly identify the *iLink-lightbulb* application when p is 0.8, Figure 29 states that only 60% all the traffic identified as the *iLink-lightbulb* application is correct identifications. Therefore, we could easily conclude that Max_Pad_Delay scheme with p set to 0.8 will be able to obfuscate the *iLink- lightbulb* application approximately 85% of the time.

Unfortunately, the high efficiency of Max_Pad_Delay scheme comes with its associated overhead. Figures 30 and 31 show the energy consumption and network bandwidth overhead respectively for Max_Pad_Delay scheme. The overhead of this scheme exceeds the overhead of the other schemes at all p values (we omit the other figures

```
Policy #1
  ID: srcIP='A', srcPort='i', dstIP='B', dstPort='j'
  CONTEXT: Location='Home' AND Time=[9PM-12AM, 6AM-9AM]
  ACTION: Padding='Normal:μ=1200,σ=100, p=1.0'
          Delay='Uniform:min=0,max=20ms'
Policy #2
  ID: srcIP='A', srcPort='k', dstIP='B', dstPort='l'
  CONTEXT: Location='Home'
  ACTION: Padding='Normal:μ=400,σ=100, p=0.6'
Policy #3
  ID: srcIP='A', srcPort='m', dstIP='D', dstPort='n'
  CONTEXT: Battery=Low OR WiFi Load=High
  ACTION: Padding='Normal:μ=1200,σ=100, p=0.6'
          Delay='Uniform:min=0,max=20ms'
Policy #4
  ID: srcIP='A', srcPort='m', dstIP='D', dstPort='n'
  CONTEXT: Battery=High OR WiFi Load=Low
  ACTION: Padding='Normal:μ=1200,σ=100, p=1.0'
          Delay='Uniform:min=0,max=20ms'
Policy #5
  ID: srcIP='A', srcPort='m', dstIP='D', dstPort='n'
  CONTEXT: Battery=High AND Location=HotSpot
  ACTION: Padding='Normal:μ=1200,σ=100, p=1.0'
          Delay='Uniform:min=0,max=20ms', IPSec
```

Figure 32: Flow policies for *traffic shaping* schemes.

due to space limitation). This is an example that *privacy preserving schemes with high efficiency will be typically associated with high overhead*. Moreover, from figures 26 and 27, we can see that the two different schemes have similar efficiency (i.e., accuracy is 60%) for the *Fitbit* application for p values of 0.8 and 0.6 respectively. Note that while the first scheme has significant overhead in terms of network bandwidth because of large p, the other scheme achieves an equivalent efficiency but with lower network overhead (lower p) and additional packet delays. *Since different schemes could have equivalent efficiency but with different overheads, policies need to be carefully designed based on the impact of these overheads on the application, user, device, and network.*

7.6.3 *PrivacyGuard* Programmability and Flexibility

We will evaluate the flexibility and programmability of PrivacyGuard in terms of the ability to provide such flexible obfuscation schemes and their efficiency under different contexts. In the following, we will refer to the policies listed in Figure 32.

We first evaluate the programmability and flexibility of PrivacyGuard. High sensitive applications such as *Fitbit* could reveal sensitive activities while the user is at home. Therefore, a policy like *Policy #1* that applies high efficient scheme (i.e., Max_Pad_Delay scheme) is used for such sensitive applications/flows. However, Figure 30 shows that such high efficient scheme incurs a 150% increase in energy consumption. Since *PrivacyGuard* has the capability to create and set different context parameters dynamically, *Policy #1* is configured to be only applied during the time periods with the sensitive activities. This flexibility in setting the policy results in a significant energy saving.

In addition, *Policy #1* and *Policy #2* show an example of the fine-grained programming ability of PrivacyGuard in configuring different policies for different flows of the same application. Different obfuscation schemes such as like Max_Pad_Delay and Norm_Pad schemes configured in *Policy #1* and *Policy #2* respectively are suitable for different flows with different traffic characteristics (i.e., periodic traffic with large packet sizes versus real time traffic).

Next, we evaluate *PrivacyGuard* ability in adapting the selected policies to the contexts changes with respect to privacy and performance. *Policy #4* is an example of a policy that applies Max_Pad_Delay scheme for high performance efficiency for sensitive applications such as the *Fitbit* application when the network load is unsaturated. Figure 28 shows that the this policy has a very high performance in obfuscating the *Fitbit* application with an accuracy as low as 15% when p is set to 1, which comes with a high network overhead of about 150% additional transmitted bytes as shown in Figure 31. However, when the network condition changes to a saturated network with high load, *PrivacyGuard* switches to apply *Policy #3* for the *Fitbit* application. Figure 31 shows that this switch adapts to the new network condition by significantly dropping the network overhead from 150% to 80% at the cost of reducing the obfuscation scheme efficiency by increasing the accuracy from 15% (high efficiency) to 40% (moderate efficiency) as shown in Figure 28.

Similarly, a change in the device context such as its battery level that changes from high to low will trigger *PrivacyGuard* to apply a similar switch from a high efficient scheme (*Policy #4*) to a low energy overhead scheme (*Policy #3*) in order to preserve the remaining battery level. Moreover, a change in the user context by moving into an insecure location from a secure location, PrivacyGuard seamlessly will enforce the IPSec tunneling scheme by switching to *policy #5* as long as the battery level of the device is high.

7.7 Conclusion and Future Work

We presented *PrivacyGuard* one of the component of *iprivacy* service, a flexible and programmable privacy-preserving framework to obfuscate the activities of sensitive IoT and mobile applications from side-channel attacks. In the next chapter, we present another component of *iprivacy*, *MirageNET* that can be used to generate fake packets and, therefore, has the potential for decreasing the precision of side-channel attacks.

CHAPTER 8

MIRAGENET - TOWARDS A GAN-BASED FRAMEWORK FOR SYNTHETIC NETWORK TRAFFIC GENERATION

8.1 Introduction

The application of GAN technology to automatically comprehend and reverse engineer the grammar of networking protocols such as TCP and UDP, or to automatically create a network traffic model of devices and mobile applications, remains to be investigated. The development of such synthetic network models might have several applications in privacy, security, and network optimization. In terms of privacy, synthetic network models of apps and devices, for example, can generate fake user activities in smart homes, reducing the accuracy of side-channel attacks for user activity inference. In security research, the synthesized network data from these models could augment training data to build effective intrusion detection systems. Moreover, real-world honeypots may be constructed utilizing fictitious devices and applications in order to better understand the attacker's behavior and interactions with these apps and devices. Furthermore, application and device synthetic network models might be utilized to replicate a variety of testing scenarios for analyzing performance concerns such as load balancing, predicting network conditions, and diagnosing network difficulties.

Creating synthetic network models of real-world devices and applications is challenging, given the black-box nature of their proprietary protocols and applications logic. Therefore, to automatically reconstruct an application or device's network model, it is necessary to automatically understand the network model's syntax and semantics and reverse engineer them without use of manual coding. While the syntax part consists of the protocol, flow, and packet structures, the semantic consists of meaningful network activity. For example, a synthetic network model for an IoT device should first generate network traffic for the ON user event before generating the network traffic for the OFF event. Thus, an efficient synthetic network model of a device or application should learn the syntax, semantic, and interdependencies among them automatically.

Given this motivation, we design, evaluate and discuss the challenges of a network packet generation framework that can automatically understand the syntax of the network packets from the raw network packets and is the first step towards realizing our vision of creating a GAN based synthetic Network traffic generation framework, *MirageNet's*, that can automatically create synthetic network models of protocols, applications, and devices. We summarize the contributions as follows:

- We design and develop *MirageNet's* initial component; a synthetic Network Packet Generator framework.
- We present a sequential byte modeling approach for network packet generation using GANs.
- We describe the challenges, limitations, and solutions towards generating synthetic network packets.
- We validate and evaluate the performance of our framework using synthesized DNS packets.

8.2 MirageNet Applications

Synthetic data generation is an emerging research topic in the machine learning domain given it can boost the performance of various machine learning algorithms [128], solve class imbalance issues when there is insufficient data for certain classes [129], and most importantly, the ability to model any unknown distribution [18]. Similarly, we believe synthetic network traffic data can enable new different applications in the network

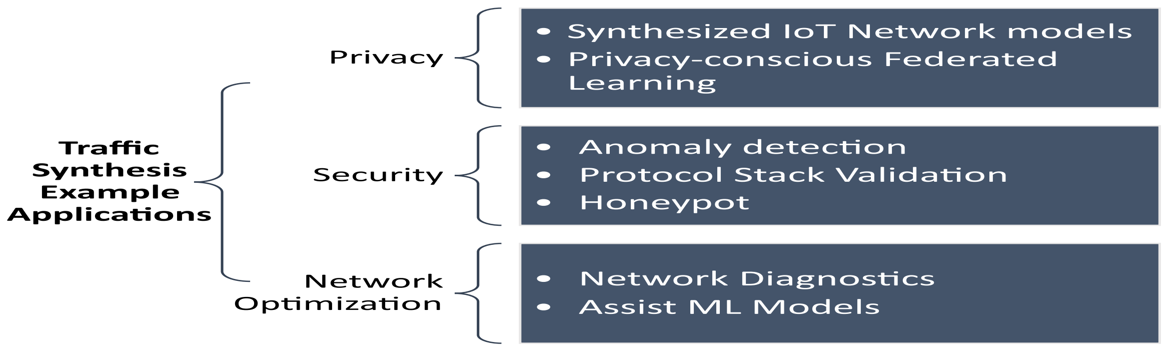

Traffic Synthesis Example Applications
Privacy
Synthesized IoT Network models
Privacy-conscious Federated Learning
Security
Anomaly detection
Protocol Stack Validation
Honeypot
Network Optimization
Network Diagnostics
Assist ML Models

Figure 33: Different research and engineering domains where synthetic network traffic would have significant impact

domain and support existing ones. Figure 33 shows the three areas of applications that we envision for synthetic network traffic: privacy, security, and network optimization.

Starting with privacy, we see multiple use cases for synthesizing network traffic such generating traffic simulating real IoT devices and using it to counter side-channel attacks to prevent the leakage of user activities in sensitive applications such as health-care[7]. Another use case is developing privacy-conscious federated learning model where instead of using real data to build network models which can lead to information leakage[130], we can train the federated learning models using synthetic data with similar data distribution.

Another major area that can utilize the synthetic network data is the security domain. For example, data augmentation using synthetic network traffic can be used to train anomaly detection systems with a low number of labeled malware samples by generating synthetic malware samples and adding it to the training data. Additionally, traffic synthesis can be used to generate packets with a wide range of characteristics to test protocol vulnerabilities in the device network stack. For example, a malformed packet can cause certain target device functionalities to fail and cause a denial of service. Building synthesized malware is another interesting application where attack patterns can be recorded by deploying "Honeypots" to capture real malware traffic and learn its properties and use the synthesized malware to build defense mechanisms that can

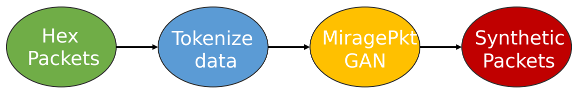


Figure 34: Packet Generation Pipeline using MiragePkt

defend against large-scale IoT-based attacks such as Mirai [11].

Finally, we also consider the area of network optimization, where synthetic network traffic simulating certain network conditions such as high congestion can be used to measure network performance. Moreover, generating scenario-specific synthetic network data can help improve the efficiency of many ML-based network optimization tools. For example, the efficiency of reinforcement learning-based models can be improved by training them under an extensive range of network conditions using GANs[128]. Given the above interesting applications, in the below section, we will discuss our framework for generating synthetic network packets as a first step towards building a complete flow generation framework.

8.3 MirageNet Packet Generation Framework

Our focus is to build a packet generation framework that can synthesize the network packets whose protocol is unknown. We choose this objective given the significance of generating synthetic data for proprietary protocols where traditional network parsing tools such as scapy and traffic generators cannot abe utilized as they fail to automatically understand the syntax of the network packet with multiple fields, values, and their corresponding correlations. Specifically, we focus on building the synthetic packet generation model using the raw data collected using a tools such as TCP dump that consists of a binary stream and does not give information about the structure of the packet. We pre-collect only the raw binary stream of the network packets as the training data and do not consider any metadata information about the structure or field values to build our protocol-agnostic packet generation framework. With this motivation, we design our packet generation as shown in Figure 34 that consists of multiple components. We discuss each of these sub-components in detail.

8.3.1 Hex stream data collection

This is the first step of our framework pipeline that performs two significant tasks, collecting the network packets and extracting raw bye streams from the network packets. The network packet collection from online or LAN networks that follow proprietary protocols is challenging, given that traditional network parsing tools such as scapy cannot identify similar packets belonging to a specific proprietary protocol. For example, collecting the DNS network packets without knowing any information about the DNS protocol is difficult. Therefore, in this component, we use Natural language processing and byte similarity-based calculations between packets to identify and filter the required raw packets by using the byte stream. However, for this work, we generate DNS packets based on a scapy script and use this component only to extract the hex stream from the packets. Designing the sub-component to filter network packets having similar protocol using byte stream is part of our future work.

8.3.2 Tokening the data

The next component of the framework is tokenization that is responsible for transforming the network byte stream data into sequences of tokens where each token is representing the byte of packet. Our intuition here is to model the packets as sequences of bytes and use GANs to understand the conditional distribution of bytes in a sequence of the packets. We first tokenize the data and performs one-hot encoding on each bytes in a sequence for creating the training data for GAN. However, our goal is to later extend this in future work with different data pre-processing methods from NLP such as word-bag, word embedding, or building a word2vec model for bytes of network packets that can better understand the correlations between the bytes of a network data.

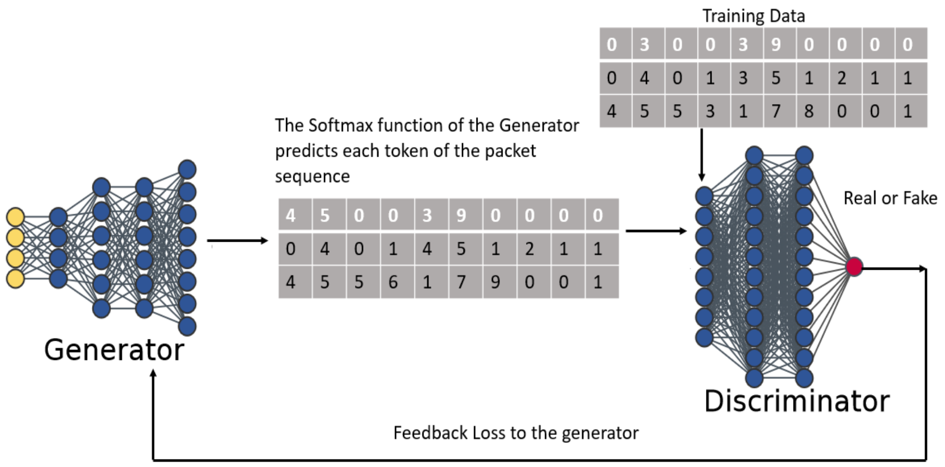


Figure 35: MiragePkT Generation Framework

8.3.3 *MiragePkt* packet generation model

Figure 35 shows the overall architecture of the *MiragePkt* GAN model, which consists of two components generator and a discriminator. The generator's objective is to generate the sequence of tokens representing a packet byte stream. The discriminator's objective is to extract the features from the generated sequence and identify whether the generated sequence belongs to the true distribution of the training data. This model training is similar to the vanilla GAN training. However, one of the significant challenges in generating sequences of bytes using vanilla GANs is its inefficiency in generating discrete data. Many major GAN architectures perform efficiently with continuous data and have shown excellent performance with image data. However, for the tasks of sequence generation GANs have performed poorly in domains such as music and text. We use a combination of 1D CNN and SoftMax functions in the generator to predict each of the tokens to form the byte sequence of a network packet. It is to be noted that, unlike a simple RNN or LSTM-based regression task that can predict a token based on the previous tokens, this GAN architecture can generate all tokens of the sequence at once, which is closer to the true distribution of the training data. The generator and discriminator models consist of ID CNN layers and are built using

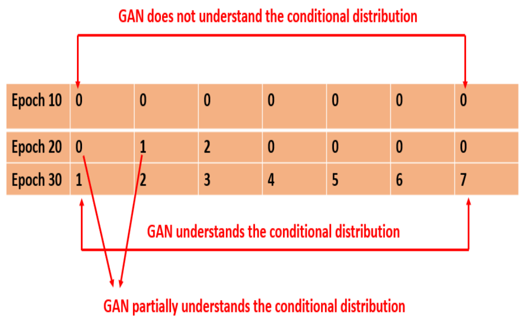

GAN does not understand the conditional distribution
Epoch 10 0 0 0 0 0 0 0
Epoch 20 0 1 2 0 0 0 0
Epoch 30 1 2 3 4 5 6 7
GAN understands the conditional distribution
GAN partially understands the conditional distribution

Figure 36: Byte Sequence generation through MiragePKT model training

the improved Wassertain GAN model training [131] that has shown improved synthetic generation for images. Very recently sequence based GANs to generate sequential data have been proposed [132, 133] that has motivated this work to model sequence of bytes as packets using GANs. Figure 36 shows the evolution in byte sequence generation of the *MiragePkt* where in the early epochs, the generator model is producing random sequences without any correlation. However, in the later epochs, the generator learns to understand the conditional distribution of the bytes in the packet and, therefore, can generate high-quality sequences at later epochs. We discuss the improvement of the packet generation during training by showing the verbose text of network packets later in the evaluation section.

8.3.4 Post processor

Finally, this last component of the framework pipeline generates the hex stream of the packet from the output of the *MiragePkt* that consists of a numerical representation of a packet sequence and then identifies if the packet is of high or low quality before sending it to the network. Determining the quality of a network packet can be measured by looking at metrics such as measuring the similarity with the training data, and measuring the randomness of the data such that the packet does not belong to the original training data or previously generated packets. We discuss some of these metrics in our evaluation section that we discuss next.

Domains
instagram.com
bing.com
google.co.uk
alibaba.com
google.com.br
google.co.in
wikipedia.org

Table 8.: Samples Domains from the Alexa dataset

8.4 Evaluation

We pick DNS packets as our use case for the modeling network packets using GANs as they follow a complex syntax grammar with multiple fields such as domain name, questions, etc., and therefore can provide better insights into the challenges of synthetic packet generation. It is to be noted that DNS packets contain variable length payloads because of the domain names and therefore in our *MiragePkt* during the tokenization step, we get the maximum length of byestreams among the training data packets to create byte sequences with the max length and append zeros for any packets less than the max length. Building packet generation for packets with dynamic payloads with variable sizes or complex conditions is much more challenging than simple deterministic network packets. We discuss next some of these challenges below.

8.4.1 Dataset Analysis and Experiment

To train *MiragePkt*, we first generate DNS packets using scapy and then use their hex stream as our raw training data. We create a python scapy [45] based script that generates DNS packets using real-world domain names as the payload. Figure 37 shows the distribution of the domains with variable length and Table 8 shows some of the sample domains from the dataset.We implement *MiragePkt* using PyTorch [84] and python and train it on a GPU-enabled machine and 32 GB memory. In the below subsections, we perform the data analysis of the training data and discuss our model evaluation with different training dataset configurations with different complexity levels.

8.4.2 Impact of Training dataset configurations

The machine learning model efficiency is highly dependent on the data pre-processing and the final input training data. Therefore, to understand the impact of the training data on *MiragePkt* models' ability to reverse-engineer the DNS network packets automatically, we create three different training datasets with different complexities, randomwithvariablelength, comwithfixedlength, comorgwithfixedlength. In the randomwithvariablelength, we pick 50000 variable-length payloads with a distribution of domain name lengths as shown in Figure 39 . Next, we create the second dataset comwithfixedlength that consits of 50000 packets with fixed domain length domain and domain suffix as .com. Finally, the comorgwithfixedlength dataset has 25000 domain names with suffix as.com and .org each. The comorgwithfixedlength configuration is to understand the impact of DNS packets with different domain suffixes on *MiragePkt*. .

From Figure 39, shows that for randomwithvariablelength configuration, *MiragePkt* performance is highly inefficient across different metrics. The *MiragePkt* major failure for this training dataset configuration is its inability to generate valid DNS packets. Table 9 shows the different packet errors found while generating the DNS packets where most of the generated packets have incorrect IP header length and UDP header length field values. *Therefore MiragePkt needs to be improved to understand unknown functions among different bytes of the packet automatically.* However, designing such a network architecture or supporting loss function is very challenging task and needs to be researched more.

To further validate our above analysis we train *MiragePkt* with comwithfixedlength

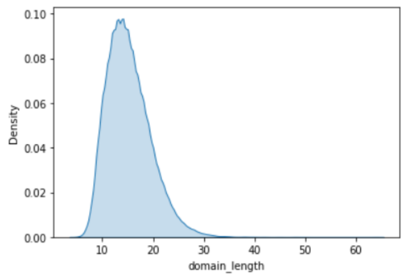


Figure 37: Domain length distribution in the dataset

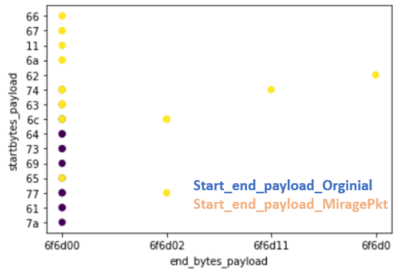


Figure 38: Features of start and end bytes

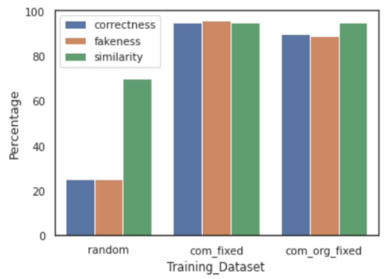


Figure 39: Performance of *MiragePkt*

and comorgwithfixedlength that have fixed length payloads but multiple domain suffixes with the intuition that model performance will increase with decrease in mathemtical dependencies of the training data . Figure 39 shows that with change of configuration from comwithfixedlength to comorgwithfixedlength the performance of the *MiragePkt* is signficantly improved with a increase in the number of correct packets from 25% to 95% and increase in fakeness from 25% to 94% . It is to be noted that we measure the fakeness of the packets by searching for the fake byte stream string in the corresponding training dataset configuration. *Therefore, MiragePkt does not memorize or overfit on the training data and generates high quality synthetic content.* Moreover, we observed that the number of domains generated for comorgwithfixedlength contains both .com and org in the generated packets. *Therefore MiragePkt was able to capture the different modes of the training data and does not fall into the mode collapse problem where the GANs only generate one mode of the distribution.*

From figure 39 there is not a significant increase the percentage of correctness or fakeness for the change of configuration from comwithfixedlength to comorgwithfixedlength where both configurations have the same fixed length of domains. *Therefore, the change in domain suffix structure does not effect MiragePkt significantly as much the mathematical dependencies introduced because of variable length domains.* It is also interesting to note that irrespective of the configuration of different configurations the similarity of the bytes of the synthetic content in all the configurations is performing

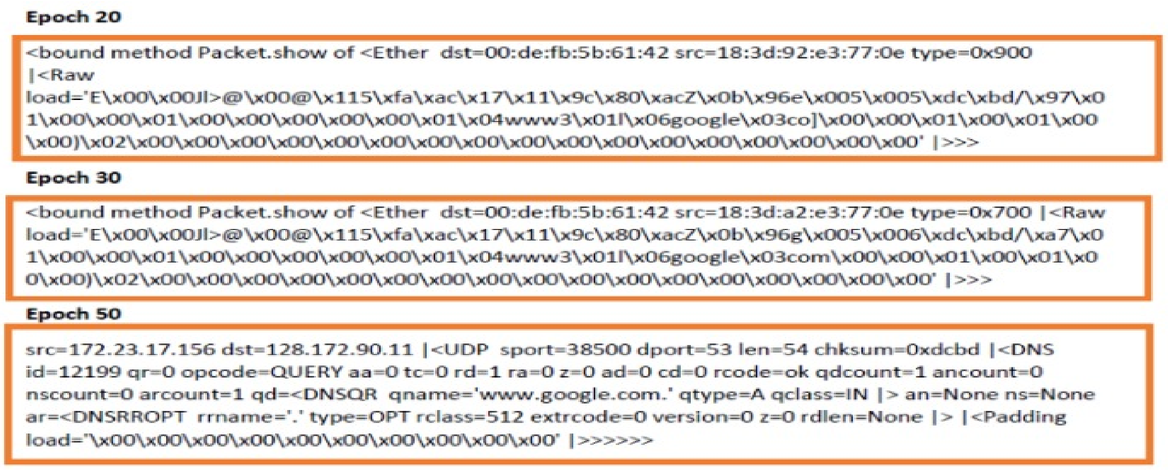
Packet Errors
Bad length value 38 > IP payload length
IPv4 total length exceeds packet length
Bad length value 39 > IP payload length
Bad length value 616 > IP payload length
Malformed Packet (Exception occurred)

Table 9.: The network packet parsing errors for generated packets

well. *Therefore, MiragePkt can generate a synthetic packet that is very similar to the original training dataset.* For calculating similarity, we performed a byte-level string to string similarity to find the max similarity of each generated packet across training dataset packets and then calculated the average of all the similarity scores. Figure 38 the scatter plot for the first and lat bytes of samples from training data *MiragePkt* showing GAN was able to completely understand the last bytes of the payload with domain suffix .com.

8.4.3 Visualization

For evaluating the synthetic packet visually similar to the process used to evaluate the quality of synthetic images, we use scapy tool to output the verbose text of the generated hex byte stream. From Figure 40 shows the improvement of packet generation during training. At epoch 20, the network packet is malformed and scapy is not able to construct a valid packet using the generated hex stream. However, at epoch 50, scapy is fully able to reconstruct the network packet with all the fields. *Thus MiragePkt can*



```
Epoch 20
<bound method Packet.show of <Ether  dst=00:de:fb:5b:61:42 src=18:3d:92:e3:77:0e type=0x900
|<Raw
load='E\x00\x00Jl>@\x00@\x115\xfa\xac\x17\x11\x9c\x80\xacZ\x0b\x96e\x005\x005\xdc\xbd/\x97\x0
1\x00\x00\x01\x00\x00\x00\x00\x00\x01\x04www3\x01l\x06google\x03co]\x00\x00\x01\x00\x01\x00
\x00)\x02\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00' |>>>
Epoch 30
<bound method Packet.show of <Ether  dst=00:de:fb:5b:61:42 src=18:3d:a2:e3:77:0e type=0x700 |<Raw
load='E\x00\x00Jl>@\x00@\x115\xfa\xac\x17\x11\x9c\x80\xacZ\x0b\x96g\x005\x006\xdc\xbd/\xa7\x0
1\x00\x00\x01\x00\x00\x00\x00\x00\x01\x04www3\x01l\x06google\x03com\x00\x00\x01\x00\x01\x0
0\x00)\x02\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00' |>>>
Epoch 50
src=172.23.17.156 dst=128.172.90.11 |<UDP  sport=38500 dport=53 len=54 chksum=0xdcbd |<DNS
id=12199 qr=0 opcode=QUERY aa=0 tc=0 rd=1 ra=0 z=0 ad=0 cd=0 rcode=ok qdcount=1 ancount=0
nscount=0 arcount=1 qd=<DNSQR  qname='www.google.com.' qtype=A qclass=IN |> an=None ns=None
ar=<DNSRROPT  rrname='.' type=OPT rclass=512 extrcode=0 version=0 z=0 rdlen=None |> |<Padding
load='\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00' |>>>>>>
```

Figure 40: Packet Visualization of *MiragePkt* using scapy

completely learn the syntax of the DNS packet with their correct field values.

8.5 Related work

Some researchers have recently worked on using GANs to build models that generate synthetic metadata of network traffic where the metadata is the high level parameters of a network flow such as number of bytes, number of packets, and flow duration [134][135]. In [135] authors used a *word2vec* embedding approach to generate the flow metadata parameters of the traffic, given an embedding approach can capture the high-level relationships among different flow metadata attributes in high dimensional space. However, their work generates the flow metadata traffic and doesn't generate the network packets. We believe their work can be extended and augmented to our packet generation model for building an initial design of a complete network traffic generation framework.

Very recently, some researchers have used GANs to generate the synthetic packets[36]. In [36] they use CNN GANs and a special encoding mechanism to generate different types of network packets such as DNS, Ping, etc. However, our work is significantly different, given we use different data prepossessing modeling using a sequence-based approach to model the bytes of a network packet. Moreover, our model design, unlike their work which uses a GAN to generate continuous values and then map it to discrete byte values, uses the softmax in the generator to predict the bytes sequences of the packet. Furthermore, unlike their work, where the data transformation is done by duplicating a byte value of a hex stream packet multiple times to achieve high similarity at the cost of increasing the dimensions of training data, our model only requires the tokenized data, therefore, is more scalable for training and packets with large byte streams. Recently authors in [136] used a sequence-based approach to generate adversarial packets, which is similar to our sequence-based modeling approach to packet generation. However, *MiragePkt* significantly differs in the model design, framework and our objective to show the challenges with packet generation using DNS packets, unlike their focus on evading IDS using adversarial packets. Moreover, *MiragePkt* is only the first component of our larger vision of building *MirageNet*.

Very very recently in [37] researchers used a style-based approach to generate a synthetic pcap file that consists of network packets and a flow. However, our work differs significantly because we focus on developing synthetic network packets when the network protocol is unknown and only look at the byte stream network packets for modeling the packet generation. Moreover, our experiments and metrics in this work are focused on exploiting the GAN capabilities to understand the limitations, challenges, and solutions to packet generation. Furthermore, as discussed in our framework section, the packet generation have multiple components whose objectives of exploiting NLP-based data processing or building post-processing tools to correct and detect the randomness of packets are very different from their work.

8.6 Discussion

In this work, we focused on the first step towards realizing *MirageNet* by developing *MiragePkt* where we were able to demonstrate the many intricacies in the packet generation process, the impact of training data, and the limitations of GANs in comprehending some of the mathematical functions. As our future work, we intend to develop a few components that improve the packet generation model.

First, we want to investigate pre- and post-processing components to aid in the preparation of better training data and to automatically fix any malformed packets with minor flaws. For example, as a pre-processing step, we can design scripts to identify non-deterministic regions in the overall training data, such as payload, and map them to any of their correlated fields, such as header length, to provide metadata that will support GAN in automatically understanding these mappings. Similarly, a post-processing tool might be created by employing a reinforcement learning (RL) agent [137] capable of learning to repair any malformed packet based on replies from a server or device that is processing the synthesized packets.

Another *MiragePkt* enhancement would be conditional generation employing conditional GANs [138], where instead of randomly generating any network packet ending in .com or .org, the *MiragePkt* may generate packets with a certain domain suffix as requested by the user during the generation stage. The next important component we intend to develop is the *MirageNet* flow generation framework in its entirety. In this framework, the model must be capable to capture the temporal dependencies between packets in a flow as well as grasp the protocol syntax. For example, in a TCP flow, the first three packets must be syn, syn-ack, and ack, in which the model must automatically capture these rules.

CHAPTER 9

CONCLUSION AND FUTURE WORK

9.1 Summary

This thesis designs and proposes a security and privacy framework to secure edge-based IoT devices against network attacks. We build and describe the different security and privacy services and their subcomponents to realize our framework. As part of countering the malware-based attacks, we design an ML-assisted security service iKnight and evaluate it with different real-world edge-based scenarios such as robustness to noise, adaptability to new devices, and lightweight model deployment. We find that Semi-Supervised GANs and other ML techniques such as Continual Learning and Knowledge distillation are efficient for building edge-based IoT classifiers. Moreover, we design a Dynamic DNN based lightweight ML model for IoT devices and evaluate their vulnerabilities and defenses.

To counter the side-channel attack for privacy inference, we designed Privacy Guard that applies dynamic privacy-preserving obfuscation schemes based on network conditions, user policy, and location. Our proposed framework can be enhanced to automatically learn the user-based policies and generate the required obfuscation schemes using machine learning techniques. Finally, we show the results of *MirageNet* where we evaluate the ability of GANs to generate synthetic network traffic. We discuss the different challenges with building the complete *MirageNet* tool and our approach. Overall, in this thesis, we show the ability and potential of our framework components to counter network-based attacks with respect to side-channel attacks and malware. However, we need to ultimately integrate all these tools to build a complete working system that we plan for our future work.

Appendix A

LIST OF PUBLICATIONS BY THE AUTHOR

This appendix presents a list of the author's published peer-reviewed publications.

A.1 Publications

Following are the list of publications:

- Uddin, Mostafa, Tamer Nadeem, and Santosh Nukavarapu. "Extreme SDN framework for IoT and mobile applications flexible privacy at the edge." 2019 IEEE International Conference on Pervasive Computing and Communications (PerCom. IEEE, 2019.)
- Nukavarapu, Santosh Kumar, and Tamer Nadeem. "Securing Edge-based IoT Networks with Semi-Supervised GANs." 2021 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops). IEEE, 2021.
- Nukavarapu, Santosh Kumar, Mohammed Ayyat, and Tamer Nadeem. "iBranchy: An Accelerated Edge Inference Platform for loT Devices." In 2021 IEEE/ACM Symposium on Edge Computing (SEC), pp. 392-396. IEEE, 2021

REFERENCES

[1] Mohd Hamim et al. “IoT based remote health monitoring system for patients and elderly people”. In: *2019 International conference on robotics, electrical and signal processing techniques (ICREST)*. IEEE. 2019, pp. 533–538.

[2] Dongsik Jo and Gerard Jounghyun Kim. “AR enabled IoT for a smart and interactive environment: a survey and future directions”. In: *Sensors* 19.19 (2019), p. 4330.

[3] Hannaneh Barahouei Pasandi and Tamer Nadeem. “Convince: Collaborative cross-camera video analytics at the edge”. In: *2020 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. IEEE. 2020, pp. 1–5.

[4] Yousef Amar et al. *An Analysis of Home IoT Network Traffic and Behaviour*. 2018. arXiv: `1803.05368 [cs.NI]`.

[5] *IFTTT Platform*. `https://platform.ifttt.com/`. Accessed: 2020-25-06.

[6] Ayyoob Hamza, Hassan Habibi Gharakheili, and Vijay Sivaraman. “IoT network security: Requirements, threats, and countermeasures”. In: *arXiv preprint arXiv:2008.09339* (2020).

[7] Noah Apthorpe et al. *Spying on the Smart Home: Privacy Attacks and Defenses on Encrypted IoT Traffic*. 2017. arXiv: `1708.05044 [cs.CR]`.

[8] TJ OConnor, William Enck, and Bradley Reaves. “Blinded and Confused: Uncovering Systemic Flaws in Device Telemetry for Smart-Home Internet of Things”. In: *Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks*. WiSec ’19. Miami, Florida: Association for Com-

puting Machinery, 2019, 140–150. ISBN: 9781450367264. DOI: 10.1145/3317549.3319724. URL: https://doi.org/10.1145/3317549.3319724.

[9] Maxime Montoya et al. “SWARD: A Secure WAke-up RaDio against Denial-of-Service on IoT Devices”. In: *Proceedings of the 11th ACM Conference on Security Privacy in Wireless and Mobile Networks*. WiSec ’18. Stockholm, Sweden: Association for Computing Machinery, 2018, 190–195. ISBN: 9781450357319. DOI: 10.1145/3212480.3212488. URL: https://doi.org/10.1145/3212480.3212488.

[10] *shodan*. https://www.shodan.io/. Accessed: 2020-25-06.

[11] Manos Antonakakis et al. “Understanding the mirai botnet”. In: *26th USENIX security symposium (USENIX Security 17)*. 2017, pp. 1093–1110.

[12] Abbas Acar et al. *Peek-a-Boo: I see your smart home activities, even encrypted!* 2018. arXiv: 1808.02741 [cs.CR].

[13] Markus Miettinen et al. “IoT SENTINEL: Automated Device-Type Identification for Security Enforcement in IoT”. In: *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)* (2017). DOI: 10.1109/icdcs.2017.283. URL: http://dx.doi.org/10.1109/ICDCS.2017.283.

[14] N. Aloysius and M. Geetha. “A review on deep convolutional neural networks”. In: *2017 International Conference on Communication and Signal Processing (ICCSP)*. 2017, pp. 0588–0592.

[15] M. Lopez-Martin et al. “Network Traffic Classifier With Convolutional and Recurrent Neural Networks for Internet of Things”. In: *IEEE Access* 5 (2017), pp. 18042–18050.

[16] N. Jmour, S. Zayen, and A. Abdelkrim. “Convolutional neural networks for image classification”. In: *2018 International Conference on Advanced Systems and Electric Technologies (IC_ASET)*. 2018, pp. 397–402.

[17] R. L. Galvez et al. "Object Detection Using Convolutional Neural Networks". In: *TENCON 2018 - 2018 IEEE Region 10 Conference*. 2018, pp. 2023–2027.

[18] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. arXiv: `1406.2661 [stat.ML]`.

[19] Augustus Odena. *Semi-Supervised Learning with Generative Adversarial Networks*. 2016. arXiv: `1606.01583 [stat.ML]`.

[20] Tim Salimans et al. *Improved Techniques for Training GANs*. 2016. arXiv: `1606.03498 [cs.LG]`.

[21] Federico Di Mattia et al. *A Survey on GANs for Anomaly Detection*. 2019. arXiv: `1906.11632 [cs.LG]`.

[22] Jeffrey Pang et al. "802.11 User Fingerprinting". In: *Proceedings of the 13th annual ACM international conference on Mobile computing and networking*. 2007, pp. 99–110.

[23] Fan Zhang et al. "Inferring Users' Online Activities Through Traffic Analysis". In: *Proceedings of the Fourth ACM Conference on Wireless Network Security*. WiSec '11. Hamburg, Germany, 2011, pp. 59–70.

[24] Q. Wang et al. "I know what you did on your smartphone: Inferring app usage over encrypted data traffic". In: *2015 IEEE Conference on Communications and Network Security (CNS)*. 2015, pp. 433–441. DOI: `10.1109/CNS.2015.7346855`.

[25] Noah Apthorpe et al. "Spying on the smart home: Privacy attacks and defenses on encrypted iot traffic". In: *arXiv preprint arXiv:1708.05044* (2017).

[26] *Open vSwitch*. `https://www.openvswitch.org/`. Accessed: 2020-25-06.

[27] Gray Stanton and Athirai A. Irissappane. *GANs for Semi-Supervised Opinion Spam Detection*. 2019. arXiv: `1903.08289 [cs.LG]`.

[28] Bruno Lecouat et al. *Semi-Supervised Deep Learning for Abnormality Classification in Retinal Images*. 2018. arXiv: `1812.07832 [cs.CV]`.

[29] Fabio Henrique Kiyoiti dos Santos Tanaka and Claus Aranha. *Data Augmentation Using GANs*. 2019. arXiv: `1904.09135 [cs.LG]`.

[30] N. N. Pise and P. Kulkarni. "A Survey of Semi-Supervised Learning Methods". In: *2008 International Conference on Computational Intelligence and Security*. Vol. 2. 2008, pp. 30–34.

[31] Bruhadeshwar Bezawada et al. *IoTSense: Behavioral Fingerprinting of IoT Devices*. 2018. arXiv: `1804.03852 [cs.CR]`.

[32] Shuaike Dong et al. *Your Smart Home Can't Keep a Secret: Towards Automated Fingerprinting of IoT Traffic with Neural Networks*. 2019. arXiv: `1909.00104 [cs.CR]`.

[33] A. Sivanathan et al. "Classifying IoT Devices in Smart Environments Using Network Traffic Characteristics". In: *IEEE Transactions on Mobile Computing* 18.8 (2019), pp. 1745–1759.

[34] A. Sivanathan, H. H. Gharakheili, and V. Sivaraman. "Inferring IoT Device Types from Network Behavior Using Unsupervised Clustering". In: *2019 IEEE 44th Conference on Local Computer Networks (LCN)*. 2019, pp. 230–233.

[35] Jorge Ortiz, Catherine Crawford, and Franck Le. "DeviceMien: Network Device Behavior Modeling for Identifying Unknown IoT Devices". In: *Proceedings of the International Conference on Internet of Things Design and Implementation*. IoTDI '19. Montreal, Quebec, Canada: Association for Computing Machinery, 2019, 106–117. ISBN: 9781450362832. DOI: `10.1145/3302505.3310073`. URL: `https://doi.org/10.1145/3302505.3310073`.

[36] Adriel Cheng. “Pac-gan: Packet generation of network traffic using generative adversarial networks”. In: *2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. IEEE. 2019, pp. 0728–0734.

[37] Baik Dowoo, Yujin Jung, and Changhee Choi. “PcapGAN: Packet Capture File Generator by Style-Based Generative Adversarial Networks”. In: *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*. IEEE. 2019, pp. 1149–1154.

[38] Auwal Sani Iliyasu and Huifang Deng. “Semi-supervised encrypted traffic classification with deep convolutional generative adversarial networks”. In: *IEEE Access* 8 (2019), pp. 118–126.

[39] Jingjing Ren et al. “Information Exposure for Consumer IoT Devices: A Multidimensional, Network-Informed Measurement Approach”. In: *Proc. of the Internet Measurement Conference (IMC)*. 2019.

[40] Mark Kliger and Shachar Fleishman. *Novelty Detection with GAN*. 2018. arXiv: `1802.10560 [cs.CV]`.

[41] Santosh Kumar Nukavarapu and Tamer Nadeem. “Securing Edge-based IoT Networks with Semi-Supervised GANs”. In: *2021 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*. 2021, pp. 579–584. DOI: `10.1109/PerComWorkshops51409.2021.9431112`.

[42] Santosh Kumar Nukavarapu, Mohammed Ayyat, and Tamer Nadeem. “iBranchy: An Accelerated Edge Inference Platform for IoT Devices”. In: *The Sixth ACM/IEEE Symposium on Edge Computing*. SEC ’21. San Jose, CA, USA: ACM, 2021. DOI: `10.1145/3453142.3493517`. URL: `https://doi.org/10.1145/3453142.3493517`.

[43] Maria Jose Erquiaga Sebastian Garcia Agustin Parmisano. *IoT-23: A labeled dataset with malicious and benign IoT network traffic (Version 1.0.0) [Data set]. Zenodo.* 2020. DOI: `http://doi.org/10.5281/zenodo.4743746`.

[44] *pkt2flow*. `https://github.com/caesar0301/pkt2flow`. Accessed: 2020-25-06.

[45] *scapy*. `https://scapy.net/`. Accessed: 2020-25-06.

[46] Chigozie Nwankpa et al. "Activation functions: Comparison of trends in practice and research for deep learning". In: *arXiv preprint arXiv:1811.03378* (2018).

[47] Alec Radford, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks". In: *arXiv preprint arXiv:1511.06434* (2015).

[48] Diederik Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *International Conference on Learning Representations* (Dec. 2014).

[49] Gido M Van de Ven and Andreas S Tolias. "Three scenarios for continual learning". In: *arXiv preprint arXiv:1904.07734* (2019).

[50] James Kirkpatrick et al. *Overcoming catastrophic forgetting in neural networks.* 2016. arXiv: `1612.00796 [cs.LG]`.

[51] Chenshen Wu et al. *Memory Replay GANs: learning to generate images from new categories without forgetting.* 2018. arXiv: `1809.02058 [cs.CV]`.

[52] Augustus Odena, Christopher Olah, and Jonathon Shlens. "Conditional image synthesis with auxiliary classifier gans". In: *International conference on machine learning.* PMLR. 2017, pp. 2642–2651.

[53] Ibrahim Ben Mustafa, Tamer Nadeem, and Emir Halepovic. "FlexStream: Towards Flexible Adaptive Video Streaming on End Devices Using Extreme SDN". In: *Proceedings of the 26th ACM International Conference on Multimedia.* MM '18. Seoul, Republic of Korea: Association for Computing Machinery, 2018,

555–563. ISBN: 9781450356657. DOI: 10.1145/3240508.3240676. URL: https://doi.org/10.1145/3240508.3240676.

[54] Mostafa Uddin, Tamer Nadeem, and Santosh Nukavarapu. “Extreme SDN Framework for IoT and Mobile Applications Flexible Privacy at the Edge”. In: *2019 IEEE International Conference on Pervasive Computing and Communications (PerCom*. IEEE. 2019, pp. 1–11.

[55] Jeongkeun Lee et al. “MeSDN: Mobile Extension of SDN”. In: *Proceedings of the Fifth International Workshop on Mobile Cloud Computing Services*. MCS ’14. Bretton Woods, New Hampshire, USA: Association for Computing Machinery, 2014, 7–14. ISBN: 9781450328241. DOI: 10.1145/2609908.2609948. URL: https://doi.org/10.1145/2609908.2609948.

[56] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[57] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556* (2014).

[58] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. *CIFAR-10 (Canadian Institute for Advanced Research)*. URL: http://www.cs.toronto.edu/~kriz/cifar.html.

[59] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. “Distilling the knowledge in a neural network”. In: *arXiv preprint arXiv:1503.02531* (2015).

[60] Jianping Gou et al. “Knowledge distillation: A survey”. In: *arXiv preprint arXiv:2006.05525* (2020).

[61] Francois Chollet et al. *Keras*. 2015. URL: https://github.com/fchollet/keras.

[62] *Raspberry Pi*. https://www.raspberrypi.org/. Accessed: 2021-16-02.

[63] *Edge TPU*. https://cloud.google.com/edge-tpu. Accessed: 2021-16-02.

[64] *Intel Neural Compute Stick 2*. https://ark.intel.com/content/www/us/en/ark/products/140109/intel-neural-compute-stick-2.html. Accessed: 2021-16-02.

[65] *Jetson Nano Developer Kit*. https://developer.nvidia.com/embedded/jetson-nano-developer-kit. Accessed: 2020-29-12.

[66] *Jetson Nano: Deep Learning Inference Benchmarks*. https://developer.nvidia.com/embedded/jetson-nano-dl-inference-benchmarks. Accessed: 2020-29-12.

[67] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: http://tensorflow.org/.

[68] *Installing TensorFlow for Jetson Platform*. https://docs.nvidia.com/deeplearning/frameworks/install-tf-jetson-platform/index.html. Accessed: 2020-29-12.

[69] *NVIDIA TensorRT*. https://developer.nvidia.com/tensorrt. Accessed: 2020-29-12.

[70] *Jetson stats*. https://pypi.org/project/jetson-stats/. Accessed: 2020-12-20.

[71] Chongxuan Li et al. *Triple Generative Adversarial Nets*. 2017. arXiv: 1703.02291 [cs.LG].

[72] Elie Alhajjar, Paul Maxwell, and Nathaniel D Bastian. "Adversarial machine learning in network intrusion detection systems". In: *arXiv preprint arXiv:2004.11898* (2020).

[73] Nuno Martins et al. “Adversarial machine learning applied to intrusion and malware scenarios: a systematic review”. In: *IEEE Access* 8 (2020), pp. 35403–35419.

[74] Maria Rigaki and Sebastian Garcia. “Bringing a gan to a knife-fight: Adapting malware communication to avoid detection”. In: *2018 IEEE Security and Privacy Workshops (SPW)*. IEEE. 2018, pp. 70–75.

[75] Yizeng Han et al. “Dynamic Neural Networks: A Survey”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021), pp. 1–1.

[76] Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. “Branchynet: Fast inference via early exiting from deep neural networks”. In: *2016 23rd International Conference on Pattern Recognition (ICPR)*. IEEE. 2016, pp. 2464–2469.

[77] Biyi Fang et al. “Flexdnn: Input-adaptive on-device deep learning for efficient mobile vision”. In: *2020 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE. 2020, pp. 84–95.

[78] Roberto G Pachecom and Rodrigo S Couto. “Inference time optimization using branchynet partitioning”. In: *2020 IEEE Symposium on Computers and Communications (ISCC)*. IEEE. 2020, pp. 1–6.

[79] Yu Cheng et al. “A survey of model compression and acceleration for deep neural networks”. In: *arXiv preprint arXiv:1710.09282* (2017).

[80] Yunchao Gong et al. “Compressing deep convolutional networks using vector quantization”. In: *arXiv preprint arXiv:1412.6115* (2014).

[81] Emily L Denton et al. “Exploiting linear structure within convolutional networks for efficient evaluation”. In: *Advances in neural information processing systems*. 2014, pp. 1269–1277.

[82] Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. "Distributed deep neural networks over the cloud, the edge and end devices". In: *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE. 2017, pp. 328–339.

[83] Mao V Ngo, Tie Luo, and Tony QS Quek. "Adaptive Anomaly Detection for Internet of Things in Hierarchical Edge Computing: A Contextual-Bandit Approach". In: *arXiv preprint arXiv:2108.03872* (2021).

[84] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

[85] Neta Zmora et al. "Neural Network Distiller: A Python Package For DNN Compression Research". In: (2019). URL: `https://arxiv.org/abs/1910.12232`.

[86] Xiaoyong Yuan et al. "Adversarial examples: Attacks and defenses for deep learning". In: *IEEE transactions on neural networks and learning systems* 30.9 (2019), pp. 2805–2824.

[87] Abdur Rahman et al. "Adversarial examples–security threats to COVID-19 deep learning systems in medical IoT devices". In: *IEEE Internet of Things Journal* (2020).

[88] Simen Thys, Wiebe Van Ranst, and Toon Goedemé. "Fooling automated surveillance cameras: adversarial patches to attack person detection". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*. 2019, pp. 0–0.

[89] Roberto G Pachecom and Rodrigo S Couto. “Inference time optimization using branchynet partitioning”. In: *2020 IEEE Symposium on Computers and Communications (ISCC)*. IEEE. 2020, pp. 1–6.

[90] Guillaume Vaudaux-Ruth, Adrien Chan-Hon-Tong, and Catherine Achard. “ActionSpotter: Deep Reinforcement Learning Framework for Temporal Action Spotting in Videos”. In: *2020 25th International Conference on Pattern Recognition (ICPR)*. IEEE. 2021, pp. 631–638.

[91] Simon Msika, Alejandro Quintero, and Foutse Khomh. “SIGMA: Strengthening IDS with GAN and Metaheuristics Attacks”. In: *arXiv preprint arXiv:1912.09303* (2019).

[92] Mihailo Isakov et al. “Survey of attacks and defenses on edge-deployed neural networks”. In: *2019 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE. 2019, pp. 1–8.

[93] Florian Tramèr et al. “Stealing machine learning models via prediction apis”. In: *25th {USENIX} Security Symposium ({USENIX} Security 16)*. 2016, pp. 601–618.

[94] Weizhe Hua et al. “Channel gating neural networks”. In: *arXiv preprint arXiv:1805.12549* (2018).

[95] Mihailo Isakov et al. “Preventing neural network model exfiltration in machine learning hardware accelerators”. In: *2018 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*. IEEE. 2018, pp. 62–67.

[96] Ali Yekkehkhany, Han Feng, and Javad Lavaei. “Adversarial Attacks on Computation of the Modified Policy Iteration Method”. In: *2021 60th IEEE Conference on Decision and Control (CDC)*. IEEE. 2021, pp. 49–56.

[97] Yoo-Seung Won et al. "Time to Leak: Cross-Device Timing Attack On Edge Deep Learning Accelerator". In: *2021 International Conference on Electronics, Information, and Communication (ICEIC)*. IEEE. 2021, pp. 1–4.

[98] Anuj Dubey, Rosario Cammarota, and Aydin Aysu. "BoMaNet: Boolean masking of an entire neural network". In: *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE. 2020, pp. 1–9.

[99] Mark Zhao and G Edward Suh. "FPGA-based remote power side-channel attacks". In: *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2018, pp. 229–244.

[100] Sanghyun Hong et al. "A Panda? No, It's a Sloth: Slowdown Attacks on Adaptive Multi-Exit Neural Network Inference". In: *arXiv preprint arXiv:2010.02432* (2020).

[101] Christian Szegedy et al. *Intriguing properties of neural networks*. 2014. arXiv: `1312.6199 [cs.CV]`.

[102] Anirban Chakraborty et al. *Adversarial Attacks and Defences: A Survey*. 2018. arXiv: `1810.00069 [cs.LG]`.

[103] Battista Biggio, Blaine Nelson, and Pavel Laskov. *Poisoning Attacks against Support Vector Machines*. 2013. arXiv: `1206.6389 [cs.LG]`.

[104] Matthew Fredrikson et al. "Privacy in pharmacogenetics: An end-to-end case study of personalized warfarin dosing". In: *23rd {USENIX} Security Symposium ({USENIX} Security 14)*. 2014, pp. 17–32.

[105] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. "Model inversion attacks that exploit confidence information and basic countermeasures". In: *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*. 2015, pp. 1322–1333.

[106] Nicolas Papernot et al. "Distillation as a defense to adversarial perturbations against deep neural networks". In: *2016 IEEE symposium on security and privacy (SP)*. IEEE. 2016, pp. 582–597.

[107] Chaowei Xiao et al. *Generating Adversarial Examples with Adversarial Networks*. 2019. arXiv: `1801.02610 [cs.CR]`.

[108] Zilong Lin, Yong Shi, and Zhi Xue. "Idsgan: Generative adversarial networks for attack generation against intrusion detection". In: *arXiv preprint arXiv:1809.02077* (2018).

[109] Neam E Boudette. "Crashes involving Tesla Autopilot and other driver-assistance systems get new scrutiny." In: *New York Times* (2021). URL: `https://www.nytimes.com/2021/06/29/business/tesla-autopilot-safety.html`.

[110] Nicholas Carlini and David Wagner. "Towards evaluating the robustness of neural networks". In: *2017 ieee symposium on security and privacy (sp)*. IEEE. 2017, pp. 39–57.

[111] C. E. Shannon. "Communication theory of secrecy systems". In: *The Bell System Technical Journal* 28.4 (1949), pp. 656–715. ISSN: 0005-8580. DOI: `10.1002/j.1538-7305.1949.tb00928.x`.

[112] Xinwen Fu et al. "Active traffic analysis attacks and countermeasures". In: *International Conference on Computer Networks and Mobile Computing (ICCNMC)*. 2003, pp. 31–39.

[113] Qixiang Sun et al. "Statistical identification of encrypted Web browsing traffic". In: *Proceedings 2002 IEEE Symposium on Security and Privacy*. 2002, pp. 19–30. DOI: `10.1109/SECPRI.2002.1004359`.

[114] S. Chen et al. "Side-Channel Leaks in Web Applications: A Reality Today, a Challenge Tomorrow". In: *2010 IEEE Symposium on Security and Privacy*. 2010, pp. 191–206. DOI: `10.1109/SP.2010.20`.

[115] Michael Backes, Goran Doychev, and Boris Köpf. “Preventing Side-Channel Leaks in Web Traffic: A Formal Approach.” In: *20th Annual Network Distributed System Security Symposium (NDSS)*. San Diego, CA, 2013.

[116] W. M. Liu et al. “PPTP: Privacy-Preserving Traffic Padding in Web-Based Applications”. In: *IEEE Transactions on Dependable and Secure Computing* 11.6 (2014), pp. 538–552. ISSN: 1545-5971. DOI: `10.1109/TDSC.2014.2302308`.

[117] H. Rahbari and M. Krunz. “Secrecy beyond encryption: obfuscating transmission signatures in wireless communications”. In: *IEEE Communications Magazine* 53.12 (2015), pp. 54–60. ISSN: 0163-6804. DOI: `10.1109/MCOM.2015.7355566`.

[118] Charles V. Wright, Scott E. Coull, and Fabian Monrose. “Traffic Morphing: An Efficient Defense Against Statistical Traffic Analysis”. In: *In Proceedings of the 16th Network and Distributed Security Symposium*. IEEE, 2009, pp. 237–250.

[119] Jeongkeun Lee et al. “meSDN: mobile extension of SDN”. In: *Proceedings of the fifth international workshop on Mobile cloud computing & services*. ACM. 2014, pp. 7–14.

[120] Mostafa Uddin and Tamer Nadeem. “TrafficVision: A Case for Pushing Software Defined Networks to Wireless Edges”. In: *IEEE 13th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*. IEEE. 2016, pp. 37–46.

[121] *Open vSwitch*. `http://openvswitch.org/`.

[122] Xueqiang Wang et al. “DeepDroid: Dynamically Enforcing Enterprise Policy on Android Devices.” In: *Ndss*. 2015.

[123] Sungmin Hong et al. “Towards SDN-Defined Programmable BYOD (Bring Your Own Device) Security.” In: *NDSS*. 2016.

[124] Makito Kano. “SeaCat: An SDN end-to-end containment architecture”. MA thesis. 2015.

[125] F. Zhang et al. “Thwarting Wi-Fi Side-Channel Analysis through Traffic Demultiplexing”. In: *IEEE Transactions on Wireless Communications* 13.1 (2014), pp. 86–98. ISSN: 1536-1276. DOI: 10.1109/TWC.2013.121013.121473.

[126] Denzil Ferreira et al. “Securacy: an empirical investigation of Android applications’ network usage, privacy and security”. In: *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*. ACM. 2015, p. 11.

[127] M. Miettinen et al. “IoT SENTINEL: Automated Device-Type Identification for Security Enforcement in IoT”. In: *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. 2017, pp. 2177–2184. DOI: 10.1109/ICDCS.2017.283.

[128] Ali Taleb Zadeh Kasgari et al. “Experienced deep reinforcement learning with generative adversarial networks (GANs) for model-free ultra reliable low latency communication”. In: *IEEE Transactions on Communications* 69.2 (2020), pp. 884–899.

[129] Vignesh Sampath et al. “A survey on generative adversarial networks for imbalance problems in computer vision tasks”. In: *Journal of big Data* 8.1 (2021), pp. 1–59.

[130] Jiahui Geng et al. “Towards General Deep Leakage in Federated Learning”. In: *arXiv preprint arXiv:2110.09074* (2021).

[131] Ishaan Gulrajani et al. “Improved training of wasserstein gans”. In: *Advances in neural information processing systems* 30 (2017).

[132] Lantao Yu et al. *SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient*. 2016. arXiv: 1609.05473 [cs.LG].

[133] Touseef Iqbal and Shaima Qureshi. “The survey: Text generation models in deep learning”. In: *Journal of King Saud University-Computer and Information Sciences* (2020).

[134] M. Rigaki and S. Garcia. “Bringing a GAN to a Knife-Fight: Adapting Malware Communication to Avoid Detection”. In: *2018 IEEE Security and Privacy Workshops (SPW)*. 2018, pp. 70–75.

[135] Markus Ring et al. “Flow-based network traffic generation using Generative Adversarial Networks”. In: *Computers Security* 82 (2019), 156–172. ISSN: 0167-4048. DOI: `10.1016/j.cose.2018.12.012`. URL: `http://dx.doi.org/10.1016/j.cose.2018.12.012`.

[136] Qiumei Cheng et al. “Packet-level adversarial network traffic crafting using sequence generative adversarial networks”. In: *arXiv preprint arXiv:2103.04794* (2021).

[137] Kai Arulkumaran et al. “A brief survey of deep reinforcement learning”. In: *arXiv preprint arXiv:1708.05866* (2017).

[138] Mehdi Mirza and Simon Osindero. “Conditional generative adversarial nets”. In: *arXiv preprint arXiv:1411.1784* (2014).

VITA

Santosh Kumar Nukavarapu worked as a research assistant at Mobile System and Intelligent communication Lab, Virginia Commonwealth University, during his Ph.D. He did his master's from Old Dominion University, Norfolk (2011), B.Tech from CMRCET College of Engineering (2009, affiliated to JNTUH University), Hyderabad, and was a software engineer before joining the Ph.D. program. He has been awarded an NSF Travel grant to attend ONAP academic summit and has served as a committee member for the Percom Artifact Technical committee and IEEE security and privacy poster jury. His research interests are generative adversarial networks, IoT security and privacy, adversarial machine learning, and edge-based machine learning models.