Fall 2020

Lightweight Deep Learning for Botnet DDoS Detection on IoT Access Networks

Eric A. McCullough
Missouri State University, Eric96@live.missouristate.edu

LIGHTWEIGHT DEEP LEARNING FOR BOTNET DDOS DETECTION ON IOT ACCESS NETWORKS

A Master's Thesis

Presented to

The Graduate College of

Missouri State University

In Partial Fulfillment

Of the Requirements for the Degree

Master of Science, Computer Science

By

Eric McCullough

December 2020

LIGHTWEIGHT DEEP LEARNING FOR BOTNET DDOS DETECTION ON IOT ACCESS NETWORKS

Computer Science

Missouri State University, December 2020

Master of Science

Eric McCullough

ABSTRACT

With the proliferation of the Internet of Things (IoT), computer networks have rapidly expanded in size. While Internet of Things Devices (IoTDs) benefit many aspects of life, these devices also introduce security risks in the form of vulnerabilities which give hackers billions of promising new targets. For example, botnets have exploited the security flaws common with IoTDs to gain unauthorized control of hundreds of thousands of hosts, which they then utilize to carry out massively disruptive distributed denial of service (DDoS) attacks. Traditional DDoS defense mechanisms rely on detecting attacks at their target and deploying mitigation strategies toward the attacker but differentiating between botnet attack traffic from normal traffic is extremely difficult, rendering mitigation strategies ineffective. An expanding body of work seeks to sidestep this difficulty by using sophisticated machine learning algorithms to detect botnet-based attacks at their source; however, many of these algorithms are computationally demanding and require specialized hardware, which is expensive, rendering them impractical. This thesis proposes a botnet detection mechanism that operates at the IoT access network. It utilizes a novel method of classifying visual representations of network activity using lightweight deep learning models. This approach is shown to be highly effective, with an average accuracy of 99.8% on a sparce dataset, perfect accuracy on an expanded dataset, and runtime latency ranging from 334 ms to 2 seconds on a Raspberry Pi.

KEYWORDS: convolutional neural networks, deep learning, distributed denial of service attacks, IoT security, long short-term memory recurrent neural networks, support vector machines

LIGHTWEIGHT DEEP LEARNING FOR BOTNET DDOS

DETECTION ON IOT ACCESS NETWORKS

By

Eric McCullough

A Master's Thesis
Submitted to the Graduate College
Of Missouri State University
In Partial Fulfillment of the Requirements
For the Degree of Master of Science, Computer Science

December 2020

Approved:

Razib Iqbal, Ph.D., Thesis Committee Chair

Ajay Katangur, Ph.D., Committee Member

Anita Liu, Ph.D., Committee Member

Lloyd Smith, Ph.D., Committee Member

Julie Masterson, Ph.D., Dean of the Graduate College

ACKNOWLEDGEMENTS

The completion of this thesis would not have been possible without the guidance of my co-chairs, Dr. Razib Iqbal and Dr. Ajay Katangur. Without the thoughtful questions which provided direction for my curiosity, the challenge to push myself toward my goals, and the limitless supply of patience they afforded me I never could have made it this far.

Thanks are also due to Dr. Jamil Saquer, Dr. Anthony Clark, Dr. Lloyd Smith, and the other faculty and staff of the Computer Science department who provided guidance and encouragement to me in the completion of this degree.

I dedicate this thesis to my family. To my mother and father, Frances and Doug McCullough, for their constant support and encouragement as I stumbled through three different majors as an undergraduate and for continuing their support through my master's degree. To my grandparents, Bob and Linda McCullough, Charles and Maxine Rogers, and Kathy and Roger Scurlock. To my great uncle, John White, and my great grandmother, Ann White. None of this could have been possible without the lessons, encouragement, and support they have offered me throughout my life.

TABLE OF CONTENTS

1 Introduction 1
- 1.1 Background 1
- 1.2 Research Motivations 2
- 1.3 Research Contribution 3
- 1.4 Overview 3

2 The Internet of Things 5
- 2.1 The Architecture of the Internet of Things 5
- 2.2 IoT Application Domains 7
- 2.3 IoT Security 9

3 DDoS Attacks and Botnets 12
- 3.1 DDoS Attacks 12
- 3.2 Botnets 15

4 Related Work 21
- 4.1 Traditional DDoS Detection Strategies 21
- 4.2 DDoS Mitigation Strategies 22
- 4.3 Machine Learning for DDoS Detection 24
- 4.4 Lightweight Deep Learning 27

5 Method 33
- 5.1 Hypothesis 33
- 5.2 System Design 33
- 5.3 Design Considerations 35

6 Experimental Results 39
- 6.1 Phase One 39
- 6.2 Phase Two 49

7 Conclusion 56

8 References 58

Appendix 65
- Smart Home Simulation 65
- Analysis Network 66
- The Internet 67

LIST OF TABLES

Table 1: Testing dataset distributions 43

Table 2: LSTM and CNN training dataset distributions 43

Table 3: Accuracy (Acc), DR, and FAR for each classifier 46

Table 4: Precision, recall, and F1 scores for each classifier 47

Table 5: Average accuracy, DR, FAR, and F1 score for each classifier 48

Table 6: Average model runtime latencies on a Raspberry Pi 48

Table 7: Number of trainable parameters for each deep learning model 48

Table 8: Smart home simulation device descriptions 52

Table 9: Distributions for the datasets gathered using the network simulator 53

Table 10: Distribution of the DDoS samples for each dataset 53

Table 11: Average runtime for creating and labelling a network traffic heatmap taken over 25,608 frames 55

LIST OF FIGURES

Figure 1: The service-oriented architecture of the IoT 6

Figure 2: A random subdomain DDoS attack. 15

Figure 3: The Mirai botnet's operations 18

Figure 4: A standard CNN convolutional layer 30

Figure 5: A MobileNet convolutional layer 30

Figure 6: An example fire module used in SqueezeNet 32

Figure 7: Proposed botnet attack detection mechanism design 34

Figure 8: Labelled heatmap examples representing a three second window of network activity 36

Figure 9: Example heatmaps used for training the CNN models 37

Figure 10: The testbed data pipeline used to test the proof-of-concept implementation of the proposed system 41

Figure 11: IoT network simulator design 50

LIST OF ALGORITHMS

Algorithm 1: Method for transforming network traffic flows into traffic windows of uniform length 42

1 INTRODUCTION

1.1 Background

The IoT is an emerging technology which integrates common devices with the Internet [1]. Once connected to a network, IoTDs can communicate with each other, web services, and applications. The IoT has been used to enable several modern conveniences, such as automated or remote user control of smart homes [2, 3, 4], efficient, low-waste infrastructure [5, 6], and real time collection and analysis of biometrics through wearable technology [7, 8, 9]. With 2 billion users reported in 2016 [10] and projections estimating that 100 billion IoTDs will be connected to the Internet by 2025 [11], the IoT has established itself as a cornerstone of the modern technological landscape. However, ensuring the security of these systems has persisted as an open issue long after their adoption [12].

IoTDs are characterized as low power devices, resulting in them receiving ineffectual or non-existent security mechanisms [12]. This has given nefarious actors on the Internet billions of easy targets for unauthorized control. IoTDs are responsible for the operations of critical systems [3, 7, 8, 9] and the transmission of sensitive data [2, 3, 13, 6], making the security of IoT systems an important issue. Attackers have gained illegitimate control of swarms of IoTDs to form what are commonly known as botnets and used them to cripple web services through distributed denial of service (DDoS) attacks [14], such as the devastating Mirai attacks of 2016 [15]. The unique topology of botnet-based DDoS attacks presents challenges that render traditional DDoS defense strategies ineffectual.

Traditional approaches to DDoS attack defense place attack detection at the victim of the attack and mitigation mechanisms at the source of the attack [16, 17, 18], but this paradigm struggles to account for key factors introduced by botnets. Botnets commonly favor application

level protocol floods [19], which are known to circumvent current DDoS detection mechanisms due to the attack traffic being sourced from seemingly legitimate IP addresses [18] and appearing indistinguishable from legitimate network correspondence [20]. Aside from the difficulty of detecting attacks sourced from botnets, deploying mitigation strategies to swarms of up to half a million attackers [15] places a prohibitively high strain on network infrastructure [18]. These difficulties have inspired a growing body of work [21, 22, 23, 24] based on the use of machine learning to detect botnet activity at the source of the attack rather than the victim.

Botnet attack detection using machine learning has been shown to be highly accurate [21, 22, 23, 24], but carries a high computational cost. Many approaches favor the use of deep learning algorithms, which have achieved state-of-the-art results across multiple application domains [25], but, while highly accurate, these algorithms often necessitate the use of GPU acceleration to ensure low runtime latency [25]. Unfortunately, the price of artificial intelligence grade GPUs places them out of reach for the average user [26, 27]. This casts doubt on the ability of deep learning-based botnet detection mechanisms to gain widespread adoption. To overcome this challenge, this thesis presents a network-level botnet attack detection system using lightweight deep learning operable on a single-board computer (SBC).

1.2 Research Motivations

Deep learning approaches to botnet attack detection are highly effective [21, 22, 23], but the computational complexity of such approaches make them impractical for access network deployment. Current approaches contain inefficiencies, such as unnecessary preprocessing [21], traffic analysis on a per-device basis [22], and analysis on a per-packet basis [23]. The motivation of this work is to develop a botnet attack detection system which provides accurate detection while scaling with the massive network sizes of the IoT.

The design of the proposed botnet attack detection system and the methods used to evaluate it are designed in accordance with the following strategies:

1) Ensure computational efficiency by:
   a) Utilizing lightweight deep learning algorithms [28, 29],
   b) Eliminating unnecessary data preprocessing.
2) Ensure scalability by:
   a) The system design not increasing in complexity with the size of the network,
   b) The system design not processing individual network packets.
3) Ensure effectiveness by testing the system with data specific to IoT botnet attack scenarios.

1.3 Research Contribution

The proposed botnet attack detection system utilizes a novel detection mechanism which operates as follows:

- The IoT network access router counts the number of ingressing and egressing packets for each host on the network over a predefined interval of time.
- At the conclusion of each interval, the packet counts are visualized as a heatmap representing network activity.
- The heatmap is classified by a lightweight convolutional neural network (CNN) as representing DDoS attack traffic or normal traffic.

1.4 Overview

The remainder of this thesis is organized as follows: the structure of the IoT in terms of architecture, application domains, and security vulnerabilities is described in Chapter 2, followed by an in-depth analysis of DDoS attacks and botnets in Chapter 3. Chapter 4 contains a literature review covering the current state-of-the-art in botnet attack detection mechanisms and lightweight deep learning models. The novel botnet attack detection mechanism is presented in Chapter 5. Chapter 6 presents the results of two phases of experimentation used to ascertain the effectiveness of the proposed system. Concluding remarks and directions for future work are

discussed in Chapter 7.

2 THE INTERNET OF THINGS

2.1 The Architecture of the Internet of Things

The term "Internet of Things" originated with Kevin Ashton in 1999 who noted,

> "If we had computers that knew everything there was to know about things- using data they gathered without any help from us- we would be able to track and count everything, and greatly reduce waste, loss and cost. We could know when things needed replacing, repairing, or recalling, and whether they were fresh or past their best." [30].

With the advances seen in that past 21 years and the mass utilization of smart technology, Ashton's vision of a world where computers integrate seamlessly into their physical environments has been realized.

When comparing the IoT with traditional network technology, the key delaminating feature is the connected "things", or IoTDs [12]. There is a high degree of variation among IoTDs in terms of size, shape, communication functionality, and power. To provide clarity among this diversity, Miorandi et al. presented a definition for IoTDs that encompasses their common characteristics. This definition outlines six defining features of an IoTD [1]:

1. It must possess some computational capacity.
2. It must be embodied by physical hardware.
3. It must possess some communication capability.
4. It must be associated with a unique identifier.
5. It must be associated with a hostname and IP Address.
6. It must possess either the ability to gather information from its physical environment (a sensor), the ability to change its physical environment (an actuator), or both.

These characteristics illuminate the greater functionality of the IoT, namely its ability to either sense or change a physical environment. When IoTDs are connected to the Internet, they

form a global and ubiquitous network which enables web services and applications to interact with the physical world [31].

Xu, He, and Li [32] provide a big-picture overview of the IoT through a service-oriented architecture (SOA). The SOA for the IoT, illustrated in Figure 1, contains four layers: the sensing layer, the networking layer, the service layer, and the interface layer [32]. Each of these layers build on the others to accomplish higher order functionality.

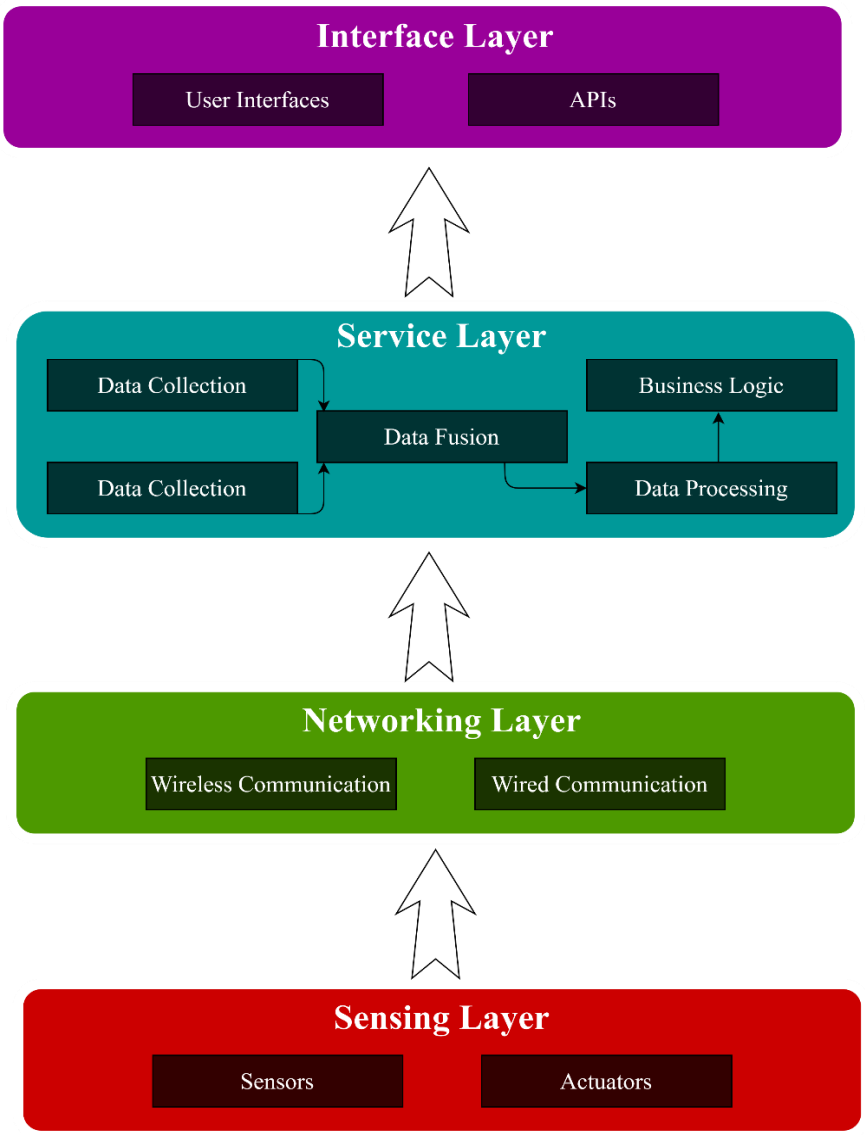

Figure 1: The service-oriented architecture of the IoT

In Xu, He, and Li's [32] SOA, each layer has clearly defined responsibilities:

- **The Sensing Layer** is comprised of sensors and actuators, which either provide information about their environment to entities in proceeding layers or allow them to change the physical world in some way.
- **The Networking Layer** consists of the protocols which enable IoTDs in the sensing layer to communicate with each other and entities in the service and interface layers.
- **The Service Layer** implements the business logic of IoT applications. It is responsible for informing services which IoTDs can provide them with data they require, assembling services together to achieve higher order functionality, providing reputation mechanisms among services to establish trust, and providing APIs and middleware which ensure services can interoperate.
- **The Interface Layer** provides APIs through which applications can communicate with each other and user interfaces which allow users to interact with the service and sensing layers.

This architecture illuminates the higher-level design of any number of IoT applications, from smart homes to industrial grids.

2.2 IoT Application Domains

IoT applications adhere to common principles in terms of design and functionality, but there is a great deal of variation among different IoT application domains. When considering the varying landscapes of different IoT domains, such as smart homes, the smart grid, and healthcare, the numerous security concerns and vulnerabilities of these systems become apparent.

2.2.1 Smart Homes. Smart homes are IoT environments where sensors and actuators are embedded in appliances, home goods, furniture, etc. By integrating these devices with the service layer and interface layer, the home environment can be automated, providing various utilities to users. Domingo [2] states that a smart home can provide an occupant with recipes tailored around their dietary requirements, taking health conditions like high cholesterol and

diabetes into account, and based on the groceries they currently have in the house. Smart homes also provide a user with the ability to control their home appliances remotely, an option which has gained commercial success recently with products such as Google Home [33] and Amazon Alexa [34].

Smart homes can facilitate significant quality of life improvements through facilitating greater independence for the elderly and those with disabilities. Ghayvat et al. [3] presented an IoT system in which they integrated sensors in a home which monitored the activity of the inhabitants. Sensor readings from the smart home were used to extrapolate the resident's wellbeing and notify caretakers if there was cause for concern. Similarly, Domingo [2] described the potential for an "object search engine" to aid blind people locate objects in their home and traverse physical obstacles.

Smart homes provide economic and environmental benefits to their users. Orsi and Nesmachnow [4] designed a system that significantly reduced the energy usage of home appliances. This was accomplished by monitoring the energy utilization of a device and optimizing its operation in achieving a desired user state. In an experiment, their system reduced the energy consumption of a water heater by nearly 40% by scheduling cycles in which the device would shut off.

2.2.2 Smart Grid. The smart grid is an emerging application of the IoT which integrates urban utility systems, such as electric grids and water systems. This allows two-way communication between the consumer and supplier of a utility, enabling increased efficiency for both parties [5]. For users, the smart grid provides increased control over the amount of money they spend each month on utilities by enabling real time updates on energy consumption [5]. On the supplier side, the smart grid enables the identification points of inefficiency and waste within

their infrastructure. The city of Tokyo integrated their water supply into the IoT and identified areas where water was being leaked. This resulted in an estimated $170 million saved annually [13]. Similarly, the smart grid enables rapid identification of the source and cause of power outages, allowing for faster repairs [6].

2.2.3 Healthcare. The IoT has been integrated into healthcare to ensure favorable medical outcomes for patients. Wearable technology can detect negative health events, such as falls [8], alerting caregivers when assistance is needed. Commercial wearable technology, such as smart watches, have recently gained popularity in allowing users to track various metrics regarding their health, like steps, heart rate, blood pressure, and the duration and quality of their sleep [9]. Dinh-Le et al. [9] state that there is opportunity for healthcare providers to use this data to better serve their patients. By making these metrics visible to healthcare providers, it would be possible for them to notice concerning trends in data gathered over a longer period than practical for regular physicals.

2.3 IoT Security

It is clear that the IoT often generates, stores, processes, and communicates sensitive data [3, 7, 8, 9]. Dinh-Le et al. [9] note that medical data is subject to legally enforceable guidelines, such as HIPAA in the United States, concerning an individual's right to privacy. Failure to properly protect the confidentiality of data in the IoT could result in legal repercussions for companies producing IoT solutions within healthcare.

The IoT also performs critical tasks [2, 3, 6, 13]. Disruptions to the operation of a smart grid could negatively impact an area's economy by preventing businesses from operating without access to necessary resources. More alarming, disruptions to the functioning of some IoT applications could endanger human users, such as systems which are critical to the care of those

with disabilities or the elderly [2, 3]. The SOA for the IoT outlines the need for trust mechanisms to be established at the service layer [32], but no such considerations are made concerning communications with the sensing layer.

Well established, mature security protocols exist for traditional computer networks, these protocols are often inapplicable to the IoT [12, 35, 36, 37, 38]. Traditional network hosts, such as desktop computers and laptops, are often seen as having virtually unlimited memory and computational power from the point of view of a security engineer, and security mechanisms in these environments can be both complex and multi-layered [12]. In contrast, the definition of an IoTD presented by Miorandi et al. [1] makes it clear that IoTDs can have extremely limited computational resources in comparison to traditional network hosts. These limitations in terms of power, memory, and CPU speed often makes traditional security measures entirely unusable on IoTDs [12, 35, 36, 38].

De Canniere, Dunkelman, and Knezevic [35] identified three design considerations that should be taken into account when creating security mechanisms for IoTDs. First, the mechanism must have a small footprint. They note that in some situations, the addition of a single logic gate could be the deciding factor between a security mechanism being included in an IoTD and the mechanism being discarded entirely. According to Beaulieu et al. [36], different IoTDs could have different factors deciding whether a mechanism's footprint is too big. Some IoTDs may present an environment in which a mechanism designed for a hardware implementation is preferred while others may require a mechanism optimized for software performance. Second, the mechanism must have low power consumption. Many IoTDs rely on batteries or other external power sources, meaning security mechanisms which significantly increase the device's power draw could be intolerable. Third, the mechanism must provide good

overall runtime efficiency. IoT systems are often time sensitive and require real-time or low latency data streams. If the security mechanism in question significantly slows the flow of data from the device, it could be rendered unusable. These factors force design engineers to leave out security mechanisms in the IoT domain or choose mechanisms that offer a compromise between security and performance [36].

The situation is further complicated by the fact that even the best security mechanisms are rendered ineffective by poor user compliance [12, 39]. Iqbal et al [12]. reported that the iBaby M3S wireless baby monitor was subject to unauthorized access simply due to the default username and password from the manufacturer both being set as “admin”. While one could hope that users would comply to best practices concerning setting and protecting their usernames and passwords, current data on the topic does not leave much room for an optimistic outlook [39].

Ironically, it is the computational resources represented by the device, no matter how limited, that attackers are after [14, 15, 40]. The concept of a botnet has recently emerged and refers to an attacker gaining remote control of mass quantities of poorly secured IoTDs, then using their combined resources to carry out tasks that would otherwise be far too complex for them individually [14]. Botnets have been able to amass armies of up to half a million infected devices, using them to launch some of the most devastating DDoS attacks in history [15].

3 DDOS ATTACKS AND BOTNETS

In 2016, the IoT security community faced what Kolias et al. [15] referred to as a "wake up call" when the Mirai botnet first became active. Mirai leveraged the poor security of default usernames and passwords reported by Iqbal et al. [12] to assemble a botnet containing an estimated 400,000 IoT devices. These devices then coordinated to launch a series of massively distributed DDoS attack against several high-profile targets, including Twitter, Netflix, Reddit, and Github, leaving them crippled for a period of nearly six hours. Making matters worse, Mirai's source code was leaked on web forums, resulting in nefarious actors producing mass quantities of derivatives and variants of Mirai, which retained its destructive potential while having enough variation in behavior to avoid detection mechanisms [15]. This event made it clear that the weak security of IoTDs is not only a liability to the IoT itself but to Internet applications at large.

3.1 DDoS Attacks

3.1.1 Attack Goals. Single origin DoS attacks have been present on the Internet for decades; however, the multi-attacker DDoS variant has seen a surge in viability recently, leading them to be considered among the most destructive cyber-attacks in use today [16, 20, 41, 42]. These attacks are launched with one of two objectives: either consume a critical resource, rendering the victim unable to service legitimate user requests, or place the victim in an invalid state that forces it to shutdown entirely. Specific attack vectors used to accomplish these goals can fall into several categories, namely network level, where the attacker tries to consume bandwidth through techniques like packet flooding, and application level, where the attacker

attempts to fool the victim into consuming critical resources like CPU cycles or memory [20].

3.1.2 Attacker Motivations. The effectiveness and ease with which attackers can launch DDoS attacks have made them a weapon of choice among malicious actors on the Internet. DDoS attacks have been launched with the intention of extortion, hacktivism, corporate sabotage, and simply as a demonstration of an attacker's capability [42]. In extreme cases, nations in conflict have carried out DDoS attacks against one another in acts of cyber warfare. Examples of this can be seen with Israel and Palestine, where both sides both launched DDoS attacks against one another [16], and North Korea, who has launched several DDoS attacks against South Korea [43]. According to Kamboj et al [43], DDoS attacks occur at an alarming rate, with 1,200 occurring each day in 2017.

3.1.3 Attack Effectiveness. Not only are DDoS attacks surging in popularity, but their destructive capability has increased by orders of magnitude in a span of only a few years. Somani et al. [42] reported that peak DDoS attack volume measured at 8 Gbps in 2004. In 2015 this number increased over 100 times, measuring at 500 Gbps. In only two years, this number would more than double, with the Mirai botnet achieving floods of over 1 Tbps [15]. The mass shift of application infrastructure to the cloud [41] has provided benefits in defending against these attacks, but the sheer volume of traffic generated by a modern DDoS attack has proved more than a match for these disadvantages, leading to the current race towards establishing reliable DDoS detection and mitigation [42].

3.1.4 Detection Avoidance Strategies. A common tactic used by DDoS attackers is to obscure their identity from the victim in order to render mitigation impossible to deploy effectively [20, 44].

3.1.4.1 Application-Level Floods. In the case of application-level flooding attacks,

DDoS attackers send valid requests to their target, depleting their CPU cycles and memory reserves by forcing them to process invalid requests. Furthermore, since the only meaningful delamination between attack traffic and benign traffic is the attackers' intent, identifying the source of an application-level flood is extremely difficult [20].

3.1.4.2 IP Spoofing. In IP spoofing, an attacker sets the header fields of their outgoing attack packets to a fraudulent IP address [44]. By having each device involved in the DDoS attack use a pool of fake IP addresses, the traffic sent from each attacker will appear to the victim as having originated from multiple sources, further complicating pinpointing the exact hosts responsible for the attack. An added benefit of this technique is that it prevents the attack volume from backfiring on the attackers, due to the fraudulent IP addresses ensuring the victim's attempt to respond to their requests will never reach them.

3.1.4.3 Reflection. A common strategy for deterring attack detection is reflection [20], in which a botnet carrying out a DDoS attack passes its traffic through middleman devices, effectively rendering the attackers invisible to the victim. This can cause mitigation strategies to be misdirected toward network hosts that are acting in earnest, damaging the reputation of the unwitting reflector and allowing the attack to continue, as the bots simply choose new reflectors to funnel traffic through.

The random subdomain attack [45], as seen in Figure 2, uses DNS servers as reflectors. Attacking bots send DNS queries to an internet service provider's (ISP) DNS server requesting an address for an invalid host on the target's domain. This DNS query is typically in the form of "*<random string>.victim.com"*. This prompts the ISP's DNS server to query the victim's authoritative DNS server recursively. The victim DNS server will be unable to resolve the query, forcing it to respond with a message indicating that the requested host does not exist.

When executed by a botnet with a sufficient number of bots, the victim's DNS server becomes overwhelmed, causing it to be too busy to respond to valid DNS queries and eventually crashing. Without their authoritative DNS server, the victim's applications will be left unreachable to users attempting to connect to their services via standard Internet protocols. Similarly, the ISP's DNS servers will eventually exhaust their available memory attempting recursive queries, until they too become unresponsive to valid user requests. While several DDoS attack variations exist, there is a consensus in the literature [16, 20, 41, 42, 43, 45] that botnets are an enabling factor behind their surge in prominence and viability.

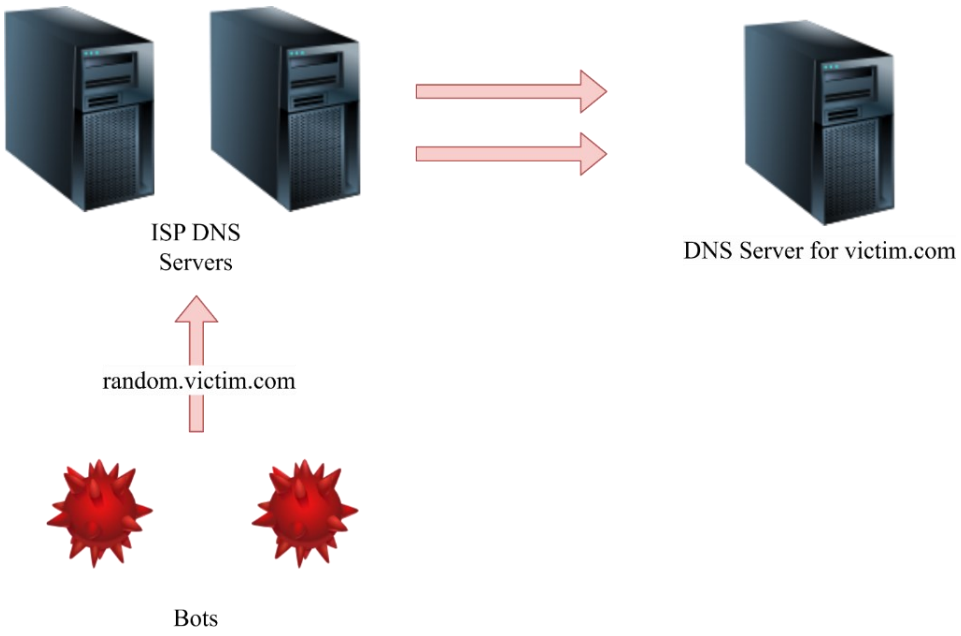

Figure 2: A random subdomain DDoS attack.

3.2 Botnets

3.2.1 Detection and Mitigation Challenges. Alieyan et al. [20] report that as many as 500 million personal devices are unknowingly infected with botnets per year, carrying an

estimated economic toll of $100 billion in 2015; yet despite the efforts of industry and academia alike, they remain a potent threat. Botnet-based DDoS attacks are particularly challenging to detect. This is because infected IoTDs appear to be valid connecting hosts due to having valid IP addresses and residing on legitimate access networks. Additionally, they favor connection-oriented attacks which appear indistinguishable from valid requests [19, 20]. Adding to the complication, botnets are often massive, with reported instances of nearly half a million bots participating in an attack while being dispersed across 164 countries [14]. The sheer number of participating attackers and the wide geographic region they are dispersed across makes attack mitigation expensive for even the most robust network infrastructures [18].

Aside from the disruption made to the target of the attack, IoT botnets often consume whatever limited processing power infected IoTDs have, rendering them unable to function within their intended purpose. In 2016, when a botnet infecting home network routers caused 900 thousand customers to lose internet connection [46]. The potential for a botnet to infect critical IoT systems, like those seen in health care [9], remains a concerning prospect.

3.2.2 Mirai: A Case Study. Researchers hoping to formulate a solution to the threat posed by botnets often make a case study of Mirai [14, 15, 47]. The prominence of this particular botnet strain in research literature is due both to its source code being publicly available [48] and historic destructive capability.

Several variations of Mirai became active shortly after the release of its source code [15]. In some cases, these variations shore up weaknesses in the original's approach, such as adding encryption between various components of the botnet [14]. In other cases, small variations are made to a known attack which enable it to skirt around defense mechanisms tailored around distinctive features of the original [41]. Many active botnets are either directly based on or draw

inspiration from Mirai, which in turn is believes to be a descendent of even older botnets [14]. Understanding its functionality can lead to insights on how current botnets operate and how their activity can be prevented.

3.2.2.1 Mirai Components. At a high level, Mirai, illustrated in Figure 3, consists of four components: the command and control server (CnC), loaders, reporters, and bots. The CnC provides the attacker a command line based [48] user interface through which they can view the botnet's status and issue commands. Bots attempt to identify other IoTDs as candidates for infection until they are ordered to attack a target through a specified DDoS method. When a candidate is identified, they send updates to reporters, who in turn provide the CnC with information on the botnet's status. The CnC can then instruct loaders to deliver the Mirai malware to the compromised IoTD [14, 15, 47]. At a more granular level, Mirai engages in activity which can be categorized as infiltration, infection, or bot operation [14].

3.2.2.2 Infiltration. Infiltration involves attempting to identify new potential IoTDs to infect. At this stage, infected bots perform port scans on hosts visible to them on the network. Mirai only scans TCP port 23 and 2323 [15], although Mirai variants and alternatives are known to target other ports [14].

Mirai specifically targets DVRs, WebIP cameras, and other IoT devices which run some distribution of Linux, primarily those with BusyBox installed [14]. If the bot successfully establishes a connection, it will attempt to start a shell on the victim IoTD via SSH or Telnet using 62 hard-coded user name and password combinations sourced from manufacturer defaults [15]. If a shell is established, the bot runs a fake command, such as "MIRAI". This step further verifies the IoTD as a viable target due to BusyBox returning a distinct error message under such circumstances, whereas known SSH and Telnet honeypots tend to respond with a help screen

[14]. If the IoTD appears to be a viable candidate, the bot will send its IP address, username, and password to a reporter [15].

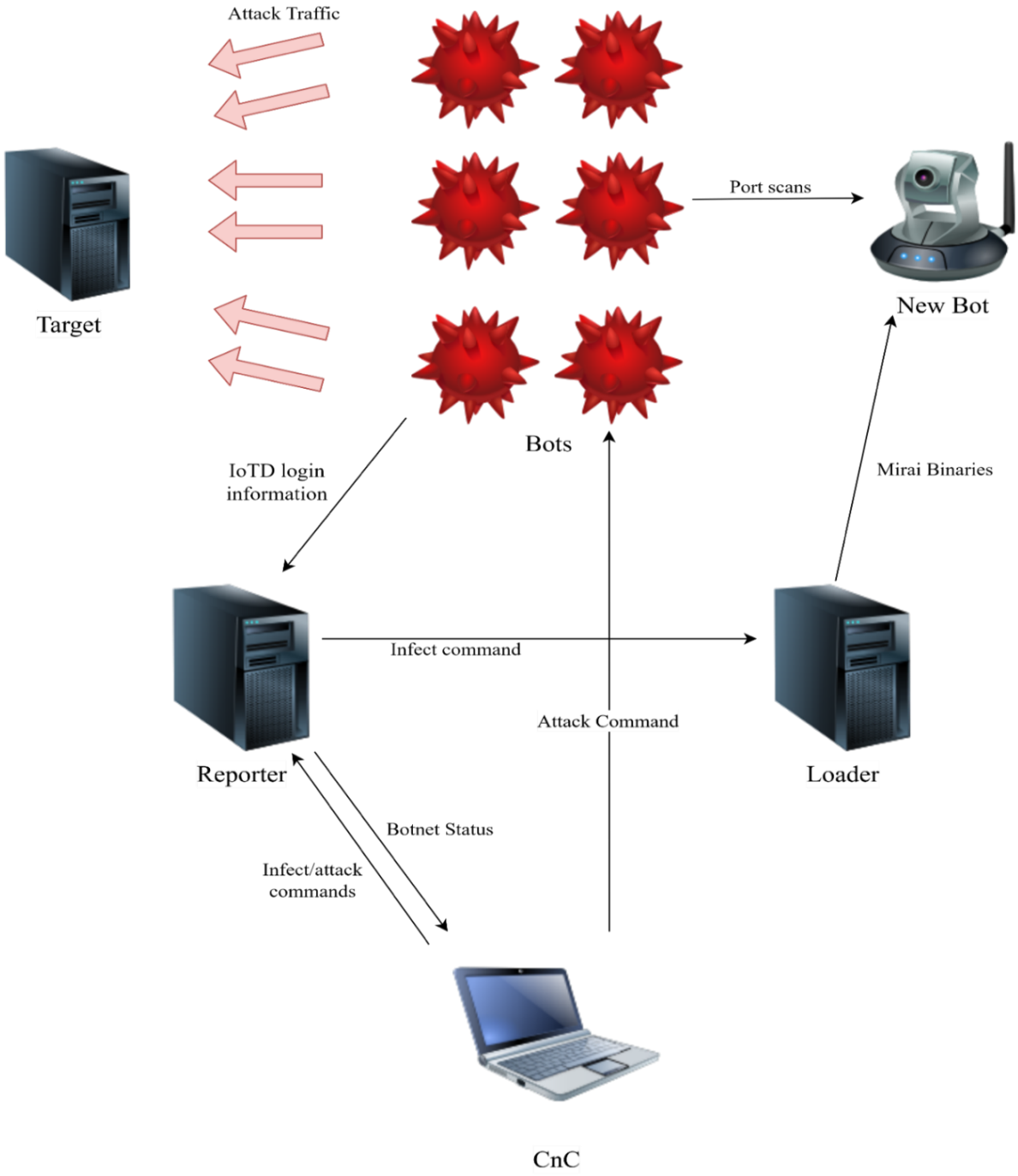

Figure 3: The Mirai botnet's operations

3.2.2.3 Infection. Infection is initiated when the attacker checks the CnC for updates, and issues commands to infect IoTD candidates known to its reporters. When the command is issued, the specified reporter will provide a loader with the login information for the target IoTD, which then logs on to the target using SSH or Telnet [15]. The loader maintains a set of low-footprint precompiled binaries [47] for the Mirai trojan targeting several ARM CPU architectures [14]. After identifying the architecture of the new bot, it will load the proper binary, if available, using Wget [49] or the trivial file transfer protocol [50].

3.2.2.4 Operation. The operation stage begins when the Mirai trojan is activated. The new bot first scans its filesystem for traces of infection by other botnets and deletes any that if finds [14]. Ironically, the bot then secures its host against the very vulnerability that allowed the infection to occur in the first place by shutting down SSH and Telnet endpoints in an effort to prevent rival botnets from stealing the device [15]. The bot then establishes a socket connection with the CnC [14] with a hostname embedded in the trojan binary. This enables the CnC to cycle IP addresses without losing its amassed army of bots [15].

The newly infected bot will then begin the cycle again by scanning the network for new IoTDs to infect using randomly generated IP addresses [47]. At any point, the CnC can issue an attack command to all available bots to commence one of ten DDoS attack variations against a specified target IP address. These attacks include, but are not limited to, HTTP, TCP, and UDP flooding [15].

3.2.3 Attack Methods and Strategies. Once a botnet has gathered enough bots, attackers can engage in high level strategies, coordinating clusters of bots and even collaborating with other botnets to cause mass disruption to Internet services. Wang et al. [19] worked with multiple ISPs to gather a dataset concerning botnet activity across the globe over a span of 200

days. From this data we can gain valuable insight into the practical operational strategies of botnets.

3.2.3.1 Attack Protocols. In terms of attack methodology, botnets seem to favor flooding attacks using connection-oriented protocols. Within this sample of botnet activity, 94% of attacks were carried out through HTTP flooding. TCP attacks were the second most regularly occurring attack, though they only accounted for slightly over 1% of the total dataset. They utilized protocol exploits like the SYN flood very rarely, instead favoring to establish full connections with their targets. UDP attacks, primarily those using reflection and amplification techniques, brought up a distant third to TCP and HTTP. The authors note that this indicates that IP spoofing is rarely used by botnets, as HTTP and TCP both require a two way connection between the attacker and the victim [19].

3.2.3.2 Botnet Collaboration. Another useful observation is that families of botnets will regularly collaborate with each other to attack a target in a way that obscures detection. Wang et al. [19] reported that botnets regularly engage in wolfpack tactics, where one group of botnets floods the target for a short period of time before stopping, allowing another group of botnets to take over. This could further complicate matters for deploying mitigation measures appropriately, as the source of the attack is constantly shifting.

4 RELATED WORK

4.1 Traditional DDoS Detection Strategies

4.1.1 Signature-Based Detection. There are two common approaches taken to detect ongoing DDoS attacks: signature-based and anomaly-based. Signature based detection typically attempts to match available data against known attack patterns. An example of this can be seen in Captcha, which presents connecting systems with a task trivial for humans to solve but surpasses the capabilities of current computer programs [51]. This approach carries the benefit of simplicity; when a new attack is discovered, unique characteristics about its activity can be identified and added to the signature database [43]. Mirai, for instance, presents distinctive network traffic signatures during its scanning and infection phases, making it a strong candidate for signature-based detection [14, 15].

Mirai also exposes the drawbacks of signature-based detection. Zero-day attacks, for which there are no known signatures, are often able to bypass signature-based mechanisms entirely [41]. Unfortunately, zero-day attacks are often accomplished by only making small tweaks to an existing attack's operation [21], as demonstrated by the prominence of Mirai variants, such as one which implements encryption between the bot and the CnC server, obscuring its identifying network traffic characteristics [14].

4.1.2 Anomaly-Based Detection. Anomaly-based detection identify attacks based on deviation from the norm. A typical implementation of this strategy sees the detection mechanism learning a systems normal state by observing it for a long period of time. When it encounters activity that appears unusual, it raises an alarm [16]. A common strategy to implement anomaly detection is to statistically model a system's operation, establishing a mathematical basis for what is normal and what is not [43]. Because this method does not use a narrowly defined

signature to detect attacks, it has the potential to identify zero-day attacks by noticing strange activity within the system it monitors.

Unfortunately, anomaly detection is also not without flaws. While a system may encounter a state that is highly unusual, that does not necessarily mean it is under attack. Anomaly detection systems by nature have a tendency to be overzealous when labelling activity as an attack, leading them to be characterized with a high false alarm rate [52].

4.2 DDoS Mitigation Strategies

A popular strategy for mitigating DDoS attacks once they are identified and the source is known is to utilize network traffic filters which prevent attack traffic from continuing past a point in the network [16]. In this paradigm, attack detection often happens at or near the victim, then filters are deployed directly to the attacker's access link [17, 18].

4.2.1 StopIt. Liu, Yang, and Lu presented a system called StopIt, which provided a framework in which filters could be deployed to an attacker's access link. In this system, designated servers were placed between routers on the Internet. If a host believed it was being attacked, it would send a request to the closest StopIt server, asking that a filter be deployed against the attacker. The server receiving the request would then identify its counterpart closest to the alleged attacker and transmit the filter request to it. The StopIt server in proximity to the attack source would then apply the filter at the appropriate access link [17].

4.2.2 ShadowNet. Bhardwaj, Miranda, and Gavrilovska presented a design for a DDoS detection and mitigation system called ShadowNet in [18]. In this design, designated ShadowNet nodes are placed at strategic locations near the network edge. Rapid communication between nodes is enabled through designated fast paths that ensure mitigation strategies can be quickly deployed in the event that a DDoS attacker is discovered. When tested on the GENI platform

with four virtual machines, an attacker, defender, and two ShadowNet nodes, they reported achieving 10x faster attack detection than comparable methods and successfully preventing 84% of attack packets from making it to the victim. It is worth noting, however, that this experimental setup does not truly replicate the structure of an IoT DDoS attack, as the attack was launched from a single origin point [18].

While ShadowNet keeps its attack detectors separated to some degree from the target host, StopIt places the responsibility for attack detection and localization entirely on the attack victim. In botnet scenarios, there are known methods for predicting the source of hundreds of thousands of attack connections, but, as Wang et al. [19] explain, these methods work best in the presence of big-picture information about the network, making them more suitable for global ISPs or countries to implement. While, intuitively, it is trivial for a host to tell it is the target of a DDoS attack, given the huge volume of traffic produced by botnets, it is another matter entirely for them to be able to pinpoint the bots responsible, especially with the prevalence of botnet strategies for hiding their location from the victim.

ShadowNet integrates its detection mechanisms with the ISP, but its mitigation strategy, like StopIt's, requires significant additions to the infrastructure of the Internet to work, casting doubt on its scalability. Recently, there has been a growing body of literature based around detecting botnet attacks at their source. This removes much of the difficulty in deploying mitigation strategies, as bots can be identified directly, eliminating the advantage of strategies like reflection or alternating groups of attackers. The downside of this approach is that many of the most distinct properties used to identify DDoS attacks, like the overall volume produced by the botnet, are obscured when only looking at a small piece of the bigger picture. This has

brought on a recent surge in approaches using machine learning techniques, as researchers hope to achieve more nuanced detection mechanisms that can operate on lower order data [53].

4.3 Machine Learning for DDoS Detection

4.3.1 Deep Feedforward Network. Diro and Chilamkurti proposed deploying a distributed deep learning model to the network edge in [21]. Their motivation behind this approach was that deep learning, which can often make more nuanced decisions than traditional machine learning methods, may be effective in detecting zero-day attacks which are commonly slight variations on known attacks. Additionally, they observed that deploying their detection model to the network edge, or fog layer, would reduce the delay in attack detection by lessening the effect of internet latency.

Diro and Chilamkurti utilized a simple deep feedforward network, what Goodfellow, Bengio, and Courville described as the "quintessential deep learning model". A feedforward network is comprised of multiple functions chained together, each operating on the output of the proceeding layer. The output of the most common deep neural network feature, called a linear layer, can be computed as:

$$y_i = \sum_{j=1} (y_{i-1})_j w_j + b \tag{1}$$

where w is the weight vector for layer i and b is the bias vector [25]. Specifically, Diro and Chilamkurti utilized a network comprised of two linear layers and a softmax activation layer [21].

To test their proposed method, they applied *1-to-n* encoding to feature vectors from the NSL-KDD dataset to transform them into input vectors to their deep learning model. Their results were favorable, with accuracy exceeding 99%. They provided an excellent analysis of the

performance of their deep learning models, providing not only raw accuracy measures, but also detection rate, false alarm rate, precision, recall, and F1 scores.

While this work has many positive qualities, it is worth noting that their deep learning model was not tested on actual edge equipment and was not designed with performance constraints in mind. Additionally, while their work was aimed at the problem of IoT DoS detection, they utilized the NSL-KDD dataset, which does not contain data from an IoT network.

4.3.2 Deep Autoencoders. Meidan et al. utilized an unsupervised deep learning technique in [22] to detect botnet activity on an IoT network. They gathered data from a testbed using actual IoTDs in their lab which they deployed the Mirai and BASHLITE botnets against. For each IoTD on the network, they then trained a deep autoencoder to identify suspicious activity.

Goodfellow, Bengio, and Courville [25] explain that autoencoders are models which attempt to map their input to their output. Meidan et al. utilized them as anomaly detectors, in which the autoencoder was trained exclusively with normal network traffic. If the autoencoder successfully recreated its input, the corresponding traffic was considered benign. Failure to recreate the input was indicative of an anomaly corresponding with botnet activity.

When the botnets were active, the autoencoders proved highly effective at labelling the traffic of a compromised device as an anomaly. Their method achieved a perfect 100% detection rate, a false alarm rate of 0.7×10^{-2}, and an average detection time of 174 ms with a standard deviation of 212 ms. While these results are indeed promising, like [21], they did not test their detection models on edge hardware. Additionally, the need to train a separate detection model for each type of device on the network makes it unlikely that this solution would scale well in diverse IoT environments.

4.3.3 Bi-Directional LSTM. Similar to the work of Meidan et al., in [23] McDermott, Majdani, and Petrovski addressed the issue of IoT botnet detection by creating a dataset using a live deployment of Mirai against an IoT testbed they built in their lab. They trained both a traditional long-short term memory (LSTM) recurrent neural network (RNN) and a bidirectional LSTM on their dataset.

According to Goodfellow, Bengio, and Courville [25], RNNs are deep learning models which operate on sequences of input, rather than fixed-sized vectors as seen in a traditional feedforward network. LSTMs are a variant of traditional RNNs which allow long-term dependencies to form between input features. This has enabled state-of-the-art performance on tasks concerning sequence processing. Bidirectional RNNs allow dependencies to be formed across an entire sequence, not just from instances occurring before the current input. Bidirectional LSTMs leverage the advances of both of these models, allowing for record breaking accuracy on tasks such as speech and handwriting recognition. McDermott, Majdani, and Petrovski utilized textual packet captures from their dataset to attempt botnet activity identification through semantic analysis of packet contents.

They found that the bidirectional model achieved slightly better accuracy than the traditional LSTM; however as with [21] and [22], it remains unclear if their approach could be reasonably deployed to edge devices. Additionally, this approach requires analysis of the contents of each packet seen coming and going from IoTDs on the network, raising concerns regarding the scalability of this detection mechanism.

4.3.4 SVM. Unlike the proceeding approaches, Bhunia and Gurusamy introduced a method for utilizing software defined networking to deploy DDoS attack detection close to the network edge in [24]. To accomplish this, the proposed deploying support vector machines

(SVM) to the control plane.

SVMs are significantly less computationally complex than deep learning models, as they only consist of a single activation function where deep models use multiple, which could indicate that they are an ideal choice for potentially constrained environments, such as onboard IoTDs. Unfortunately, Goodfellow, Bengio, and Courville note that SVMs have lagged behind the accuracy of deep models for over fourteen years [25]. There could be a tradeoff between performance and accuracy when comparing deep models and SVMs, but the degree of this tradeoff is uncertain. On one hand, if SVMs can perform satisfactorily in terms of accuracy, the performance gains they offer could trump the higher accuracy offered by deep models. On the other hand, if deep models can be shown to operate with acceptable latency in constrained environments, their high degree of accuracy would make them the undeniable choice for this application domain.

4.4 Lightweight Deep Learning

Deep learning models have gained success in a large part due to an explosion in model size, according to Goodfellow, Bengio, and Courville [25]. Single features within deep models are typically unable to learn tasks or data features which are particularly useful, but when used in concert they are able to far surpass the accuracy of conventional methods; however, this increase in model size was only possible due to advancements in computing power made available by general purpose GPUs [25].

GPUs, once utilized exclusively for rendering three-dimensional graphics, have recently seen an increase in demand which manufacturers have not been able to keep up with. In 2018, the New York Times reported that the renaissance of artificial intelligence, mainly enabled by deep learning models, and cryptocurrency mining led to shortages in GPU supplies. Shortages

inevitably led to inflated prices, with a 15% increase reported in only six months [26]. These shortages have shown no sign of relenting, as Nvidia announced earlier this year that supply for their new 30 series GPUs is not expected to stabilize until 2021 [54]. With prices starting at $819 per month for the use of a single GPU enabled server [27], the usage of deep learning models at an industrial scale carries a high monetary cost.

The cost and shortages of deep learning hardware along with a desire to enable deep learning in constrained environments has led to a recent interest in lightweight deep learning models. In this thesis, two lightweight convolutional neural networks (CNN) are utilized: MobileNet [28] and SqueezeNet [29].

Goodfellow, Bengio, and Courville [25] explain that CNNs are naturally suited to processing data laid out in a grid-like format. This has made them a natural choice for image processing, where pictures are represented as two-dimensional grids of pixels. The high degree of success CNNs have achieved in analyzing visual data is commonly attributed to the current interest in deep learning; however, even so far as deep models go, CNNs are notoriously large, with modern implementations containing several million trainable parameters [25].

Howard et al. and Iandola et al. developed MobileNet and SqueezeNet respectively to lower the computational cost of image processing using CNNs. While they share a similar goal, the two models differ in which performance metrics they prioritize and what mechanisms they utilize to achieve their desired optimization.

4.4.1 MobileNet. MobileNet, introduced in [28], is named as such due to the authors' goal of targeting mobile environments, such as smart phones. While their approach yielded a model with fewer parameters than comparable CNNs, their primary goal was to reduce the latency from receiving an input to calculating the output of the network. They accomplished this

by carefully considering what features to use for their network.

The standard convolutional layer seen in most CNNs, illustrated in Figure 4, uses a standard 3x3 convolution to both extract features from the input matrix and combine them into a more condensed output. This output is then normalized and passed through the ReLU activation function to compute the input to the next layer. In contrast, MobileNet's convolutional layers, seen in Figure 5, utilize a depth-wise separable 3x3 convolution to achieve feature extraction followed by a standard 1x1 convolution to down sample the extracted features. While this approach would appear at first glance to utilize more computations since twice as many convolution operations are used, depth-wise convolutions use eight to nine times more efficient than standard convolutions, yielding significant performance gains for MobileNet. Further, placing priority on 1x1 convolutions has the effects of decreasing the model's latency, as highly optimized implementations of 1x1 convolutions are available, and decreasing the size of the model since 1x1 convolutions utilize fewer parameters than their higher dimensioned counterparts.

Howard et al. provided an in-depth analysis of the tradeoffs between model performance and accuracy by comparing MobileNet's accuracy on the ImageNet dataset to other prominent CNNs. They found their model was within 1% of VGG16's accuracy despite having 32 times fewer parameters and utilizing 27 times fewer computations. Likewise, it showed an increase in accuracy over GoogleNet, despite being slightly smaller and utilizing 2.5 times fewer computations [28].

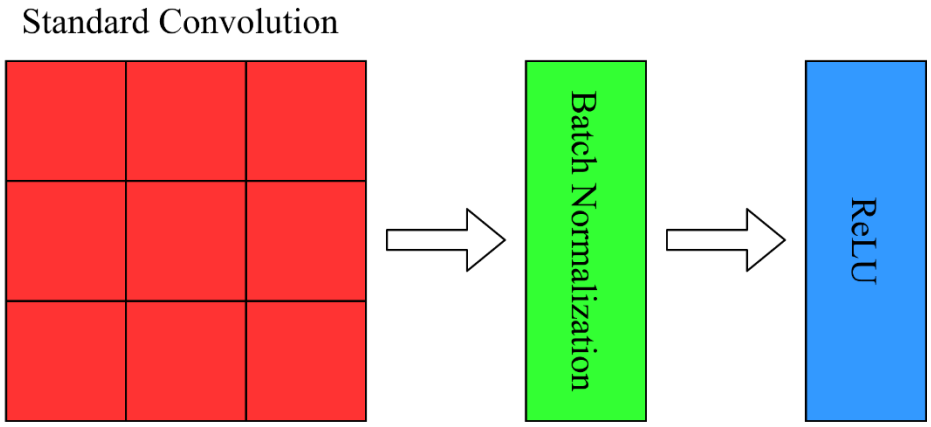

Figure 4: A standard CNN convolutional layer

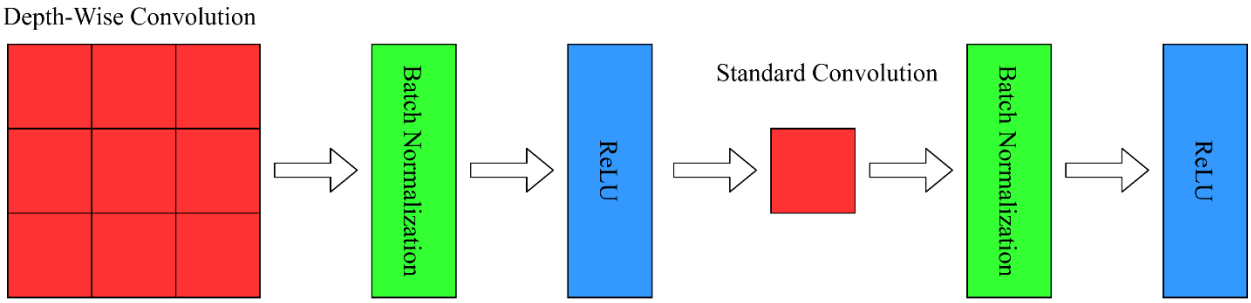

Figure 5: A MobileNet convolutional layer

4.4.2 SqueezeNet. SqueezeNet, described in [29], was designed to target applications, such as autonomous vehicles, where CNNs are integrated into embedded systems. In contrast to MobileNet, Iandola et al. prioritized making their model as small as possible by replacing 3x3 filters with 1x1 filters wherever they could and decreasing the number of input channels to 3x3 filters where they are necessary. 1x1 filters utilize nine times fewer parameters than 3x3 filters and the number of parameters to any given layer is a product of the number of input channels it receives, rendering SqueezeNet comparatively tiny in relation to other CNNs.

The basic building block of SqueezeNet is the fire module, consisting of a squeeze convolutional layer and an expand convolutional layer followed by a ReLU activation function,

as shown in Figure 6. The squeeze layer consists of exclusively 1x1 filters whereas the expand layer contains a mixture of 1x1 filters and 3x3 filters. This design ensures the prominence of 1x1 filters that the authors aimed for while also reducing the dimensionality of any input received by the 3x3 filters. Each fire module in the network contains three tunable dimensions: S_{1x1}, E_{1x1}, and E_{3x3}. These variables define the number of 1x1 convolutions in the squeeze layer, the number of 1x1 convolutions in the expand layer, and the number of 3x3 convolutions in the expand layer respectively.

SqueezeNet is remarkably small, with four times fewer parameters than MobileNet, fifty times fewer parameters than AlexNet, a CNN which largely inspired its design, and a memory footprint of 0.5 MB. Despite its reduced complexity, SqueezeNet proved highly accurate on the ImageNet dataset, matching or exceeding AlexNet's accuracy. Unfortunately, neither the authors of SqueezeNet or MobileNet compared their respective models in terms of accuracy.

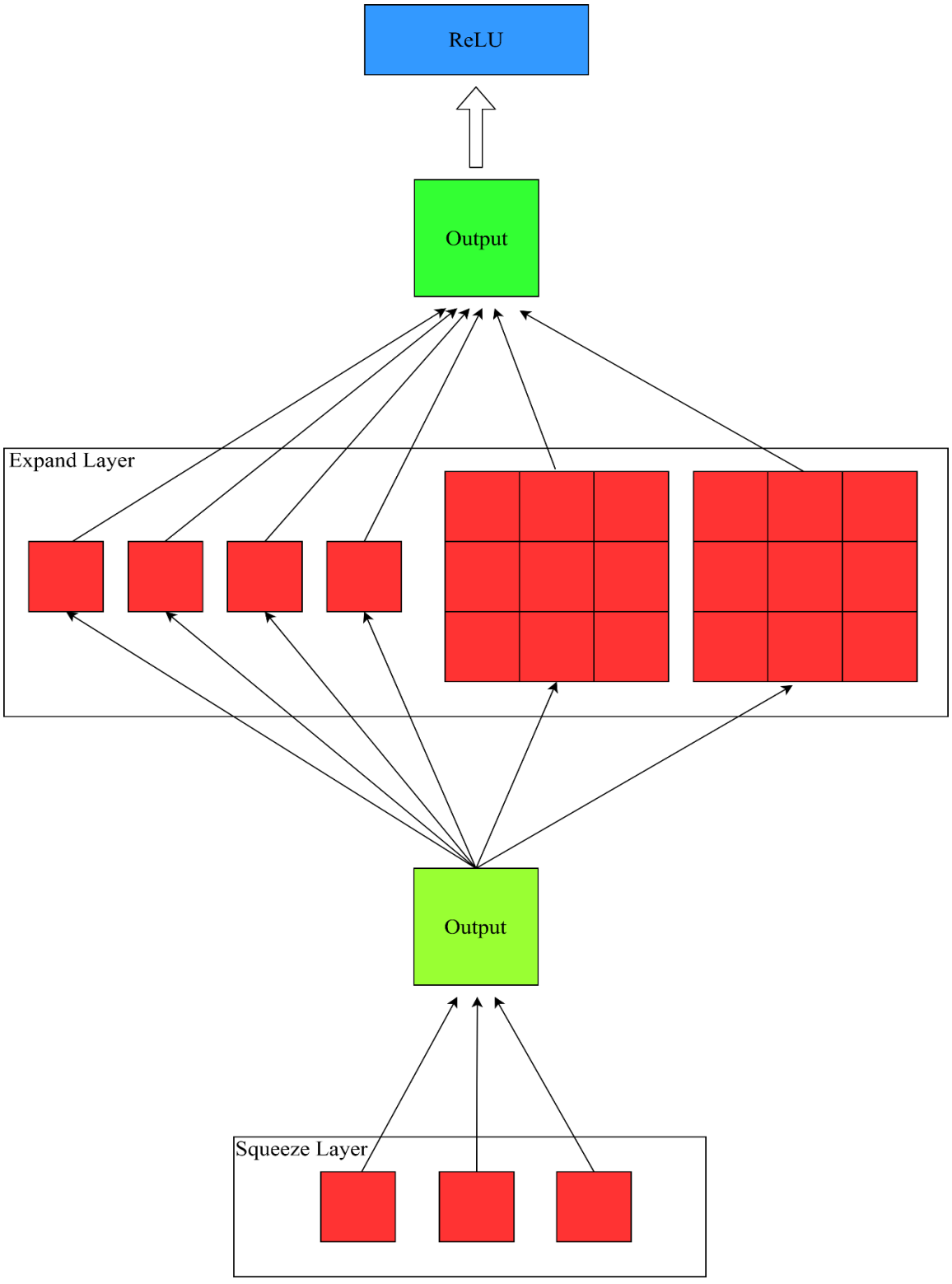

Figure 6: An example fire module used in SqueezeNet

5 METHOD

5.1 Hypothesis

The method for IoT botnet attack detection proposed in this work is motivated by the advances in botnet attack detection using machine learning [21, 22, 23, 24] described in the previous chapter. While the deep learning-based approaches [21, 22, 23] show particularly high accuracy, the computational complexity of these algorithms calls into question their viability on hardware commonly available on access networks. Inversely, the approach utilizing an SVM [24] may be viable on standard hardware, but this computational efficiency comes at the cost of reduced accuracy. With the advancements seen in lightweight deep learning [28, 29] this work proposes that the tradeoff of accuracy for efficiency is no longer necessary. SqueezeNet and MobileNet are both significantly smaller than traditional CNNs; however, they are larger than traditional machine learning classifiers such as an SVM. With a greater number of trainable parameters, these lightweight CNNs are likely to significantly outperform traditional machine learning approaches in terms of accuracy. Further, their designs prioritize minimizing runtime latency on traditional CPUs, indicating that traditional machine learning methods may not offer a significant performance advantage over them. The following section describes a botnet attack detection system which achieves accuracy comparable to state-of-the-art deep learning while retaining low runtimes, even on an SBC.

5.2 System Design

The proposed IoT botnet attack detection system is comprised of three components: a data collection module, a data formatting module, and a data analysis module. At a high level, the data collection module gathers information about the network. The data formatting module

reformats the raw data into input which the data analysis module can operate on. Finally, the data analysis module uses a lightweight CNN model to determine if the gathered information indicates that IoTDs on the network are engaging in a DDoS attack. Figure 7 illustrates the proposed attack detection system.

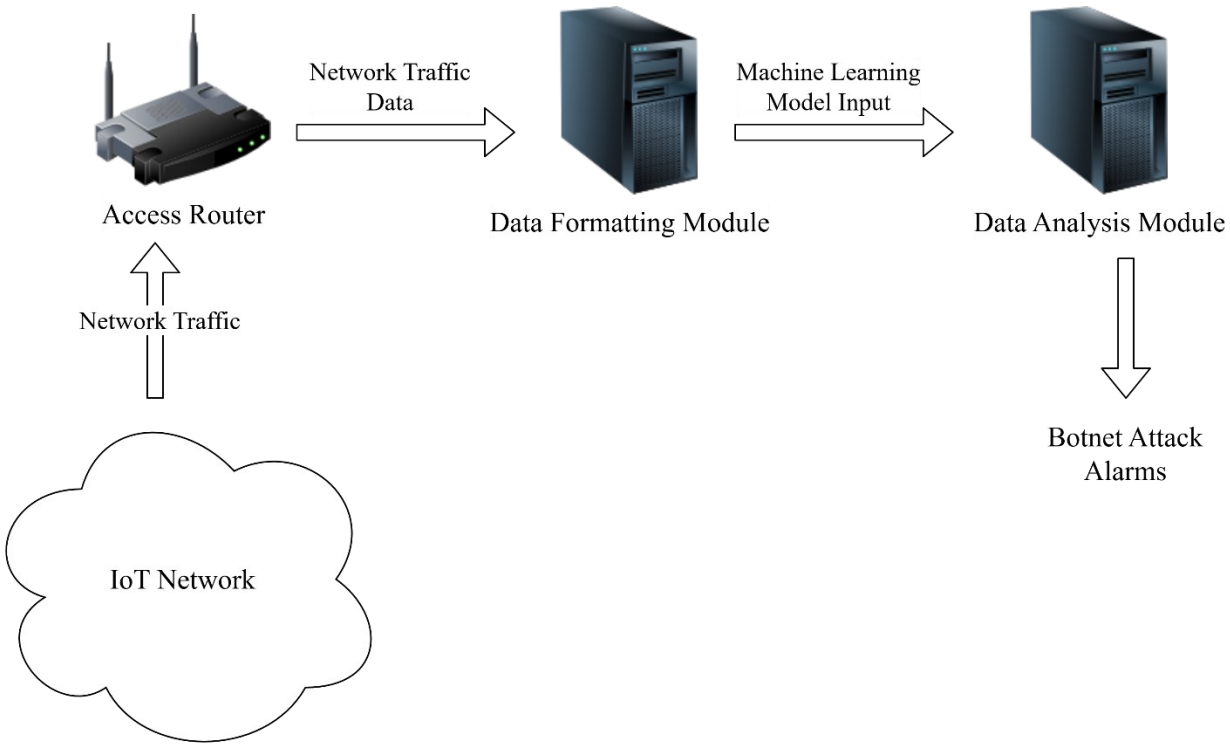

Figure 7: Proposed botnet attack detection mechanism design

5.2.1 Data Collection. The data collection module is integrated into the IoT access router as a subsystem. To avoid routing performance bottlenecks, it maintains constant runtime complexity and linear memory complexity. Concretely, it maintains a count of packets seen coming and going from each IoTD on the network and exposes these aggregated packet counts via an HTTP endpoint that the data formatting module makes requests to. When a request is received, it returns the current values for each host and resets the counters.

5.2.2 Data Formatting. The data formatting module requests data from the data collection module at regular intervals and reformats it into input for the lightweight CNN classifier. CNNs are traditionally used in computer vision applications [25], and both MobileNet and SqueezeNet expect input in the format of a 255x255x3 matrix representing an image with three color channels. This thesis presents a novel method for visualizing network traffic as graphical heatmaps, which can serve as the visual input the CNNs were designed for.

As seen in Figures 8 and 9, the heatmaps represent the volume of traffic seen coming and going from each IoT host on the network using color. Visually cooler colors, such as black, purple, and blue, indicate relatively low amounts of traffic. Warmer colors like orange, red, and white indicate gradually increasing packet counts. Each row of the heatmap represent a single IoT host on the network, whereas the columns indicate the ingressing and egressing packet counts for a particular host, respectively. For clarity, the heatmaps in Figure 8 are fully labelled; however, the heatmaps presented to the CNNs had extraneous visual information, such as axis labels, removed, leaving only the colored cells of the heatmaps, as shown in Figure 9.

5.2.3 Data Analysis. The data analysis module is hosted on an SBC and hosts an HTTP endpoint. The data formatting module sends heatmap images to the analysis module via post requests. The images are classified by the lightweight CNN classifier and the classification is sent in response to the initial request.

5.3 Design Considerations

5.3.1 Performance. In [21], the authors utilized the precomputed features of the NSL-KDD dataset, which are computationally taxing to compute and unnecessary in the context of

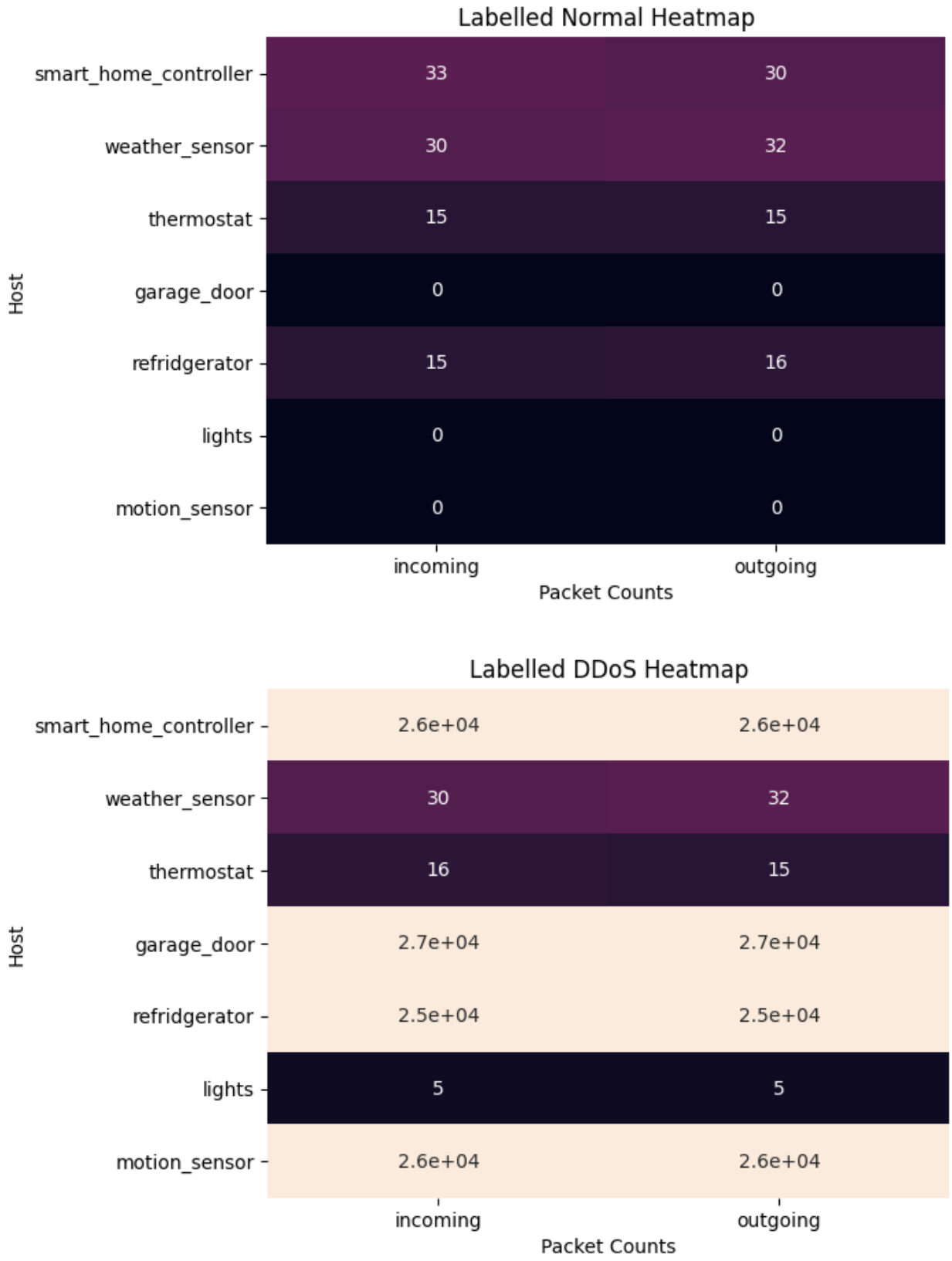

Figure 8: Labelled heatmap examples representing a three second window of network activity

Figure 9: Example heatmaps used for training the CNN models

deep learning. Deep learning algorithms use multiple layers to extract gradually more complex features from data [25]. This reduces the need to engineer pre-computed features to serve as model input, instead allowing data to be fed directly into the model. The only preprocessing used in the proposed attack detection mechanism is the creation of graphical heatmaps, which is a constant time operation.

5.3.2 Scalability. The approaches presented in [22] and [23] operated on raw network data but required models to be trained for each host in the former case and extensive analysis of individual packets in the later. This calls the scalability of both methods into question in the context of massive IoT networks. The proposed detection mechanism maintains ingressing and egressing packet counts for each host on the network, making the memory complexity of data collection $O(n)$, where n is the number of hosts on the network. The only operation necessary to gather this data is addition, making the runtime complexity $O(1)$. The image size of a heatmap is always 255x255x3, regardless of the number of hosts on the network, ensuring its runtime complexity would remain the same regardless of network size.

5.3.3 Effectiveness. The authors of [21] utilized the NSL-KDD dataset for the

evaluation of their approach. A shortcoming of this dataset is that it does not contain attack specific to IoT botnet scenarios. As discussed in Section 3.2.3, botnets rely on specific attack patterns and strategies that make them more difficult to detect than traditional DDoS attacks. To ensure the proposed detection mechanism is effective, IoT botnet-specific data is used to evaluate the system in Chapter 6.

6 EXPERIMENTAL RESULTS

The evaluation of the novel botnet detection mechanism proposed in Chapter 5 takes place in two phases. The goal of phase one is to establish the effectiveness of the proposed system in a proof-of-concept testbed using an IoT botnet dataset. Aside from accuracy measures, model performance is also established in terms of runtime on an SBC at this stage.

The purpose of phase two is to further investigate the effectiveness of the system in an environment better resembling a live IoT environment using a simulated smart home network. The smart home simulation is used to test the proposed system in attack scenarios where the number of infected IoT bots on the network is varied. While phase one utilized and SBC, phase two establishes runtime performance for the system on standard computing hardware.

6.1 Phase One

6.1.1 Experimental Setup. To ascertain the effectiveness of the proposed detection model, a proof-of-concept testbed is developed. The BoT IoT dataset provided by Koroniotis et al. in [55] is utilized as a source of test data. This dataset provides CSV files containing network flow statistics gathered from a simulated IoT network tailored toward botnet research. It covers numerous botnet-based attacks including data theft, probing, and denial of service. Within the latter category, it covers single-origin DoS attacks and DDoS attacks with data from UDP, TCP, and HTTP floods for both variants.

6.1.1.1 Data Pipeline. The BoT IoT dataset provides ingressing and egressing packet counts for each IoT device on the simulated network developed by Koroniotis et al. [55], but it does so in relation to network flows. These flows vary wildly in duration, with some lasting less

than a second while others last well nearly an hour. Figure 10 illustrates the data pipeline that transforms the BoT IoT network flows into collections of packet counts representing discrete, non-overlapping windows of network traffic.

6.1.1.1.1 The Data Transformation Module. The data transformation module is responsible for transforming the network flow CSVs of the BoT IoT dataset into discrete, non-overlapping network traffic windows. As outlined in Algorithm 1, each network flow is split into uniform time intervals and flows occurring within the same window of time are aggregated together.

The length of a network traffic window must be short enough to minimize the delay between the beginning of an attack and its detection but long enough to allow the model to run on the formatted data without developing a backlog. According to Wang et al. [19], botnet-based DDoS attacks usually last within one of three time distributions: 6-7 minutes, 20-40 minutes, and 2-3 hours. At this stage of experimentation, 20 seconds is selected as the length of a network traffic window since it allows for early detection while virtually guaranteeing backlogs do not build up.

6.1.1.1.2 The Data Formatting Module. The data formatting module, described in Section 5.2.2, is implemented to take the output from the data transformation and produce testing and training datasets for the data analysis module. The effectiveness of the proposed detection mechanism is compared to an SVM, utilized in [24], and an LSTM, as seen in [23]. A heatmap dataset is created for the lightweight CNNs while a dataset of packet count arrays is used for the LSTM and SVM.

6.1.1.1.3 Dataset Descriptions. Four distinct datasets are formed based on traffic protocol: a TCP traffic set, a set containing TCP and HTTP traffic, a UDP traffic set, and a set

containing all protocols. Since the proposed detection mechanism is only aware of network-layer data, HTTP and TCP should appear indistinguishable to it; however, a separate TCP set is utilized to verify this assumption.

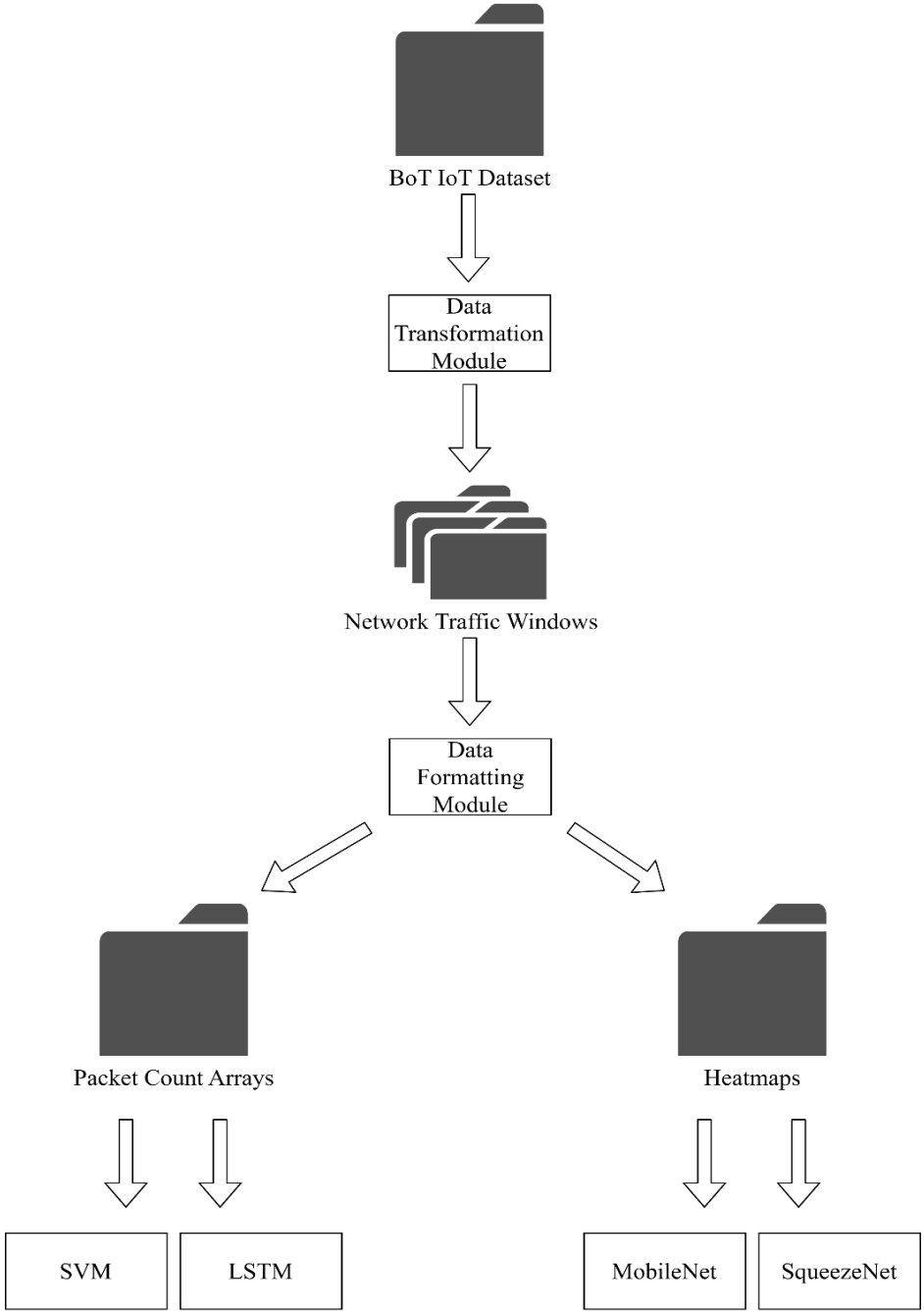

Figure 10: The testbed data pipeline used to test the proof-of-concept implementation of the proposed system

```
Input: Array of flows sorted by start time
current_frame = []
frame_start = flows[0].start_time
frame_end = frame_start + frame_length
for each flow in flows do
    if startsAndEndsInFrame(flow) then
        current_frame.append(flow)
    else if startsInFrameEndsAfter(flow) then
        percent_in_frame = findOverlap(flow, frame_start, frame_end)
        // Packets sent in current frame
        src_pckts_in = flow.src_pkts * percent_in_frame
        dst_pckts_in = flow.dst_pkts * percent_in_frame
        // Packets sent after current frame
        src_pckts_out = flow.src_pkts * (1 − percent_in_frame)
        dst_pckts_out = flow.dst_pkts * (1 − percent_in_frame)
        // Make new flow containing data outside current window
        newFlow = newflow(flow.addr, src_pckts_out, dst_pkcts_out)
        flows.insert(newFlow)
        flow.src_pckts = flow.src_pckts_in
        flow.dst_pckts = flow.dst_pckts_in
        current_frame.append(flow)
    else
        write(current_frame)
        current_frame = []
        frame_start = frame_end
        frame_end += frame_length
    end if
end for
```

Algorithm 1: Method for transforming network traffic flows into traffic windows of uniform length

For each machine learning model, the datasets are split evenly between testing and training data. For the CNNs and LSTM, the training data is separated further, with a training and validation dataset receiving 30% and 20% of the total set, respectively.

Deep learning models have a tendency to grow a bias towards classification classes that are over-represented in their training data, therefore the training datasets for the LSTM and

CNNs are down sampled to provide even representation of normal and attack traffic. Table 1 shows the distribution of the testing datasets used for all four models, while Table 2 show the distribution for the down sampled training data presented to the deep learning models.

The datasets in Table 2 are extremely small as far as deep learning training data goes. The dataset containing all protocols is 73x smaller than the smallest dataset used in [23], 707x smaller than the smallest dataset used in [21], and 2,170x smaller than the smallest dataset used in [22].

Table 1: Testing dataset distributions

Dataset	**Normal Samples**	**DDoS Samples**
UDP	2037	62
TCP	2030	57
TCP + HTTP	2030	89
All traffic	4067	150

Table 2: LSTM and CNN training dataset distributions

Dataset	**Normal Samples**	**DDoS Samples**
UDP	36	36
TCP	34	34
TCP + HTTP	53	53
All traffic	89	89

6.1.1.1.4 The Data Analysis Module. The models are trained and tested using the datasets

described in the previous section. The models are trained then evaluated on the testing dataset. Their performance is then evaluated by recording the runtime of the trained models on a Raspberry Pi.

6.1.2 Tools. The data transformation module was implemented using C++ while the rest of the code base is written in Python and makes extensive use of its rich machine learning and data science ecosystems. Pandas [56] is used to process and manipulate the network traffic windows produced by the data transformation module. NumPy [57] is used to create and store the array datasets used for training the LSTM and SVM. Network traffic heatmaps are created using Seaborn [58] and Matplotlib [59]. The deep learning models are developed with PyTorch [60], and implementations of MobileNet [61] and SqueezeNet [62] available through PyTorch Hub are utilized. Finally, the SVM is implemented with scikit-learn [63].

6.1.3 Model Accuracy Analysis. As mentioned in the literature review chapter, Diro and Chilamkurti provided a thorough understanding of their model's accuracy. Their accuracy measures were applied in this work to achieve a thorough understanding of the performance of the proposed detection mechanism.

Six accuracy measures were utilized in [21]: accuracy, which measures the number of correctly labelled inputs; detection ratio (DR), which measures the ratio of attack examples correctly labelled; false alarm rate (FAR), which measures the number of benign traffic samples which were incorrectly identified as an attack; precision, the measure of how many examples labelled by the detection mechanism as attack traffic were actually attack traffic; recall, the measure of how many instances of attack traffic were correctly identified; and the F1 score, which is used to measure how well the network balances precision and recall. Given the following values:

- **True Positive (TP):** the number of training examples correctly labelled as attack traffic
- **True Negative (TN):** the number of training examples correctly labelled as benign traffic
- **False Positive (FP):** the number of training examples incorrectly labelled as attack traffic
- **False Negative (FN):** the number of training examples incorrectly labelled as normal traffic

these accuracy metrics can be calculated as follows:

$$Accuracy = \frac{(\mathrm{TP} + \mathrm{TN})}{(\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN})} \tag{1}$$

$$DR = \frac{\mathrm{TP}}{(\mathrm{TP} + \mathrm{FN})} \tag{3}$$

$$FAR = \frac{\mathrm{FP}}{(\mathrm{TN} + \mathrm{FP})} \tag{4}$$

$$Precision = \frac{\mathrm{TP}}{(\mathrm{TP} + \mathrm{FP})} \tag{5}$$

$$Recall = \frac{\mathrm{TP}}{(\mathrm{TP} + \mathrm{FN})} \tag{6}$$

$$F1\ Score = \frac{2\mathrm{TP}}{(2\mathrm{TP} + \mathrm{FP} + \mathrm{FN})} \tag{7}$$

Table 3 gives the results for model accuracy, DR, and FAR taken over each dataset. Table 4 provides the values observed for precision, recall and F1 scores. In Table 5, the average accuracy, precision, recall, and F1 scores for each model taken over all datasets are provided for convenient comparison of model performance.

6.1.4 Model Runtime Performance. To ascertain the performance of the models when running on an SBC, each model was evaluated based on runtime on a Raspberry Pi 4 [64] with four gigabytes of RAM and 1.5 gigahertz processor. Table 6 gives the average time to label a traffic frame taken over 4,219 frames.

Table 3: Accuracy (Acc), DR, and FAR for each classifier

Model	Dataset	Acc(%)	DR(%)	FAR(%)
SVM	All traffic	99.5	94.0	0.3
	TCP	99.9	98.2	0.0
	TCP + HTTP	99.9	97.8	0.0
	UDP	99.7	90.3	0.0
SqueezeNet	All traffic	99.8	98.7	0.2
	TCP	99.8	98.2	0.1
	TCP + HTTP	97.8	98.9	0.1
	UDP	99.7	100.0	0.3
MobileNet	All traffic	99.8	100.0	0.2
	TCP	99.8	100.0	0.1
	TCP + HTTP	99.9	100.0	0.1
	UDP	99.7	100.0	0.3
LSTM	All traffic	99.7	97.4	0.2
	TCP	99.9	100.0	0.1
	TCP + HTTP	99.8	98.9	0.1
	UDP	99.9	96.8	0.0

6.1.5 Discussion. Based on the results shown in Table 4, the SVM performed the worst

out of all models used. MobileNet outperformed SqueezeNet in every metric except FAR. The LSTM achieved a better accuracy, FAR, and F1 score than MobileNet, but did not achieve its perfect DR. Based on a tradeoff of between a 2% improved detection rate against a 0.08% increased risk of false alarms, MobileNet's performance arguably surpasses the LSTM in terms of detection accuracy. However, it should be noted that the CNNs faced a severe disadvantage due to the size of the dataset.

Table 4: Precision, recall, and F1 scores for each classifier

Model	Dataset	Precision (%)	Recall (%)	F1 Score (%)
SVM	All traffic	91.0	94.0	92.5
	TCP	100.0	98.2	99.1
	TCP + HTTP	100.0	97.8	98.9
	UDP	100.0	90.3	94.9
SqueezeNet	All traffic	95.5	98.7	97.1
	TCP	94.9	98.2	96.6
	TCP + HTTP	97.8	98.9	98.3
	UDP	91.2	100.0	95.4
MobileNet	All traffic	95.0	100.0	97.4
	TCP	96.6	100.0	98.3
	TCP + HTTP	97.8	100.0	98.9
	UDP	89.9	100.0	94.7
LSTM	All traffic	94.8	97.4	96.1
	TCP	96.6	100.0	98.3
	TCP + HTTP	96.7	98.9	97.8
	UDP	98.4	96.8	97.6

The accuracy of deep learning models is attributed to their size; however, large models were only made possible recently due to an increase in dataset sizes [25]. In the presence of sparse training data, smaller models will often outperform larger ones, since they have fewer trainable parameters to adjust.

Table 5: Average accuracy, DR, FAR, and F1 score for each classifier

Metric	SVM	SqueezeNet	MobileNet	LSTM
Acc (%)	99.8	99.3	99.8	99.8
DR (%)	95.1	99.0	100.0	98.3
FAR (%)	0.1	0.2	0.2	0.1
F1 Score (%)	96.4	96.9	97.3	97.5

Table 6: Average model runtime latencies on a Raspberry Pi

Model	Average runtime (seconds)	standard deviation
SVM	0.3×10^{-3}	5.9×10^{-5}
SqueezeNet	3.3×10^{-2}	5.3×10^{-2}
MobileNet	2.0	1.5×10^{-2}
LSTM	0.1×10^{-2}	1.0×10^{-4}

Table 7 shows that SqueezeNet and MobileNet are several thousand times bigger than the LSTM. It is reasonable to assume that an increase in the amount of training data would lead to substantial increases in the accuracy attainable by SqueezeNet and MobileNet.

Table 7: Number of trainable parameters for each deep learning model

Model	Number of parameters
SqueezeNet	1,235,496
MobileNet	3,504,872
LSTM	214

In terms of runtime performance, the SVM outpaced its competitors followed closely by the LSTM. While slower, the runtimes of MobileNet and SqueezeNet fell well below the length

of a network traffic window, indicating that much faster DDoS detection is possible if the duration of network traffic windows are decreased.

While these preliminary results are promising as a proof-of-concept, there are several ways in which our understanding of this system's effectiveness could be improved:

- Increasing the size of the dataset,
- Factoring the creation of the heatmap into the runtime of the system,
- Experimenting with shorter network traffic windows,
- Add variation to the number and identity of attackers in the DDoS data. In the BoT IoT dataset these factors are always the same. In an actual IoT botnet scenario, the number of infected IoTDs will vary.

Phase two builds on the results of phase one by addressing these issues.

6.2 Phase Two

6.2.1 Experimental Setup. An IoT smart home simulator, illustrated in Figure 11, is utilized to further ascertain the effectiveness of the proposed botnet attack detection system. It replicates the operation stage of Mirai described in Section 3.2.2.4. Simulated IoTDs and an access router form the IoT network comprise the smart home simulation. To provide isolation between normal network activity and data collection, the data formatting and analysis modules run in a separate analysis network. The analysis network also contains a database instance for persistent long-term storage. Because botnet CnC servers and DDoS attack targets are not on an IoT bot's local network, these components are hosted in a third network, representing the Internet. The simulation testbed shown in Figure 11 is described in further detail in the Appendix.

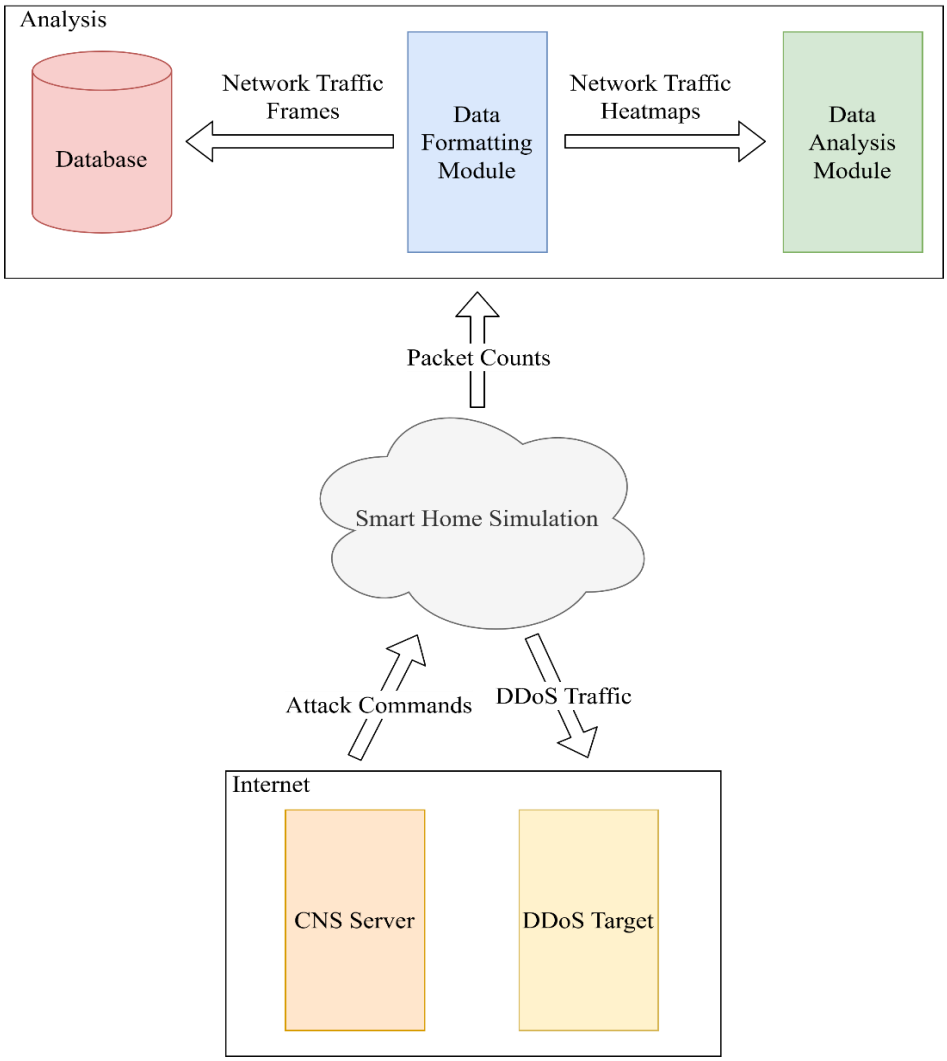

Figure 11: IoT network simulator design

6.2.1.1 Interactive Mode. In interactive mode, the CnC server and data formatting module are available for user interaction. The CnC server hosts a web interface which allows a user to select any combination of IoTDs from the smart home network and instruct them to attack the target for a specified period. Additionally, a user may connect to the data formatting module via SSH to see the packet counts for each network traffic window and the label given to each window by the data analysis module.

6.2.1.2 Data Collection Mode. In data collection mode, the network simulation is run

for a specified duration while network traffic windows are recorded. At the beginning of data collection, a traffic window size and traffic classification are specified by the user. If normal traffic is being collected, the smart home simulation is runs uninterrupted while the data formatting module requests packet counts from the data formatting module at the conclusion of each traffic window. The packet counts for each traffic window are written to the database for later usage.

6.2.1.3 Testing Mode. Testing mode allows the collection of runtime performance and accuracy metrics. To collect runtime performance, the data formatting module queries the database for all available network traffic data, formats each window into a network traffic heatmap, and sends the heatmap to the data analysis module for classification. The time that elapses between the data formatting module beginning to create a heatmap to it receiving a classification from the analysis module is recorded for each window. To collect accuracy metrics, each heatmap is written to the filesystem of the computer hosting the simulation. This allows the analysis module to run on batches of inputs, decreasing the amount of time necessary to train the network and gather accuracy metrics.

6.2.1.4 Smart Home Network Description. The smart home simulation consists of seven IoT devices, described in Table 8. The devices operate in the subscriber and publisher architecture, where subscribers receive information or commands from a set of publishers. To ensure dynamic fluctuations of network traffic when the simulation is running normally, some devices publish information at random intervals.

To replicate the computational limitations of IoTDs, each simulated device was allocated only a fraction of the simulation host's CPUs. For the purposes of this experiment, the simulation is run on a server with dual Intel Xeon E5-2440 processors, each providing six cores at 2.4

gigahertz each [65]. According to Adebija et al. [66], the current state-of-the-art mid-range processor for IoT devices provides cores running at one gigahertz each. Based on this information, each IoTD was allocated a maximum of half the processing time of a CPU.

Table 8: Smart home simulation device descriptions

Device	Classification	Traffic Description
Smart Home Controller	Subscriber/Publisher	Subscribes to updates from the weather sensor and refrigerator. Publishes new set temperatures to the thermostat on random intervals. Publishes commands to the garage door to open or close.
Weather Sensor	Publisher	Publishes randomly generated weather information to the smart home controller and thermostat every second.
Thermostat	Subscriber	Subscribes to commands from the smart home controller to set a new internal temperature. Subscribes to updates from the weather sensor to know whether it should use heating or cooling based on exterior temperatures.
Garage Door	Subscriber	Subscribes to open and close commands from the smart home controller
Refrigerator	Publisher	Publishes interior temperature to smart home controller ever second.
Lights	Subscriber	Subscribes to commands to turn on or off from the motion sensor.
Motion Sensor	Publisher	Publishes commands to turn on or off the lights at random intervals.

6.2.1.5 Dataset Description. Datasets are gathered using both three and five-second-long traffic windows. Table 9 shows the size and distribution of both datasets while Table 10 shows the variation of the DDoS samples in each dataset in terms of the number of attacking bots from one to seven, the later representing all IoTDs on the network participating in an attack.

Table 9: Distributions for the datasets gathered using the network simulator

Interval Length	Normal Examples	DDoS Examples
3 Seconds	11,576	14,032
5 Seconds	7,043	8,500

Table 10: Distribution of the DDoS samples for each dataset

Number of Attackers	3 Second Dataset	5 Second Dataset
1	1,951	1,121
2	2,012	1,185
3	2,031	1,214
4	1,991	1,218
5	2,014	1,229
6	2,023	1,249
7	2,010	1,284

Traffic windows of three and five-seconds long are selected to minimize detection latency based on the lightweight CNN's runtimes on the Raspberry Pi seen in Table 6. Because MobileNet's runtime latency is slightly over two seconds on average, three seconds is selected to

ensure fluctuations in runtime do not contribute to a bottleneck. The proposed detection mechanism views data gathered over a period of time, so it is possible that making the length of the traffic window could affect the accuracy of the proposed detection mechanism. A traffic window that is overly short could present a slice of network activity too small for the CNN to draw consistent conclusions. Traffic windows of five seconds are also gathered to determine if the length of the traffic window has any effect on the accuracy of the detection mechanism.

6.2.2 Tools. Docker [67] is used to create the virtual hosts and networks which comprise the simulation. The smart home IoTDs are implemented in Rust [68] to ensure low resource consumption. DDoS attack traffic are generated with the Nping packet generation tool [69]. The ip route [70] is used to redirect all traffic to and from the IoTDs through the access router. Packet counting is achieved with iptables [71]. Aggregated packet counts are made available to the data formatting module through an HTTP endpoint implemented using the Rocket framework [72]. HTTP endpoints for the CnC server and data analysis module are implemented with the Flask framework [73].

6.2.3 Results. Table 11 gives the average runtime for processing input, including creating the heatmap, across the entire dataset described in Table 10. This data was recorded when the data formatting module and the data allocation module were each allocated a single CPU core from the simulation's host. While these CPU cores are certainly more powerful than the Raspberry Pi's, but better represent standard, non-constrained network hardware without GPU acceleration. Both SqueezeNet and MobileNet achieved perfect accuracy when classifying the network traffic collected from the network simulation.

6.2.4 Discussion. The results of this trial are encouraging. In the simulated IoT network, data collection did not impose a bottleneck on regular routing activities. The detection

mechanism's runtime latency continued to stay within acceptable ranges when the process of formatting data into graphical heatmaps was included in runtime analysis. Further, in the presence of an expanded dataset, the proposed detection mechanism achieved perfect accuracy when classifying TCP flooding attacks. As stated previously, HTTP and TCP attacks account for over 95% of botnet-based DDoS attacks. Since the proposed system treats application-level protocols the same as their underlying network-level protocol, this indicates excellent performance in common botnet attack scenarios.

Table 11: Average runtime for creating and labelling a network traffic heatmap taken over 25,608 frames

Model	**Average runtime (seconds)**	**Standard deviation**
SqueezeNet	1.3×10^{-1}	3.3×10^{-2}
MobileNet	1.6×10^{-1}	3.5×10^{-2}

7 CONCLUSION

In this thesis, I presented a novel botnet-based DDoS detection mechanism which uses lightweight CNNs to classify graphical representations of network traffic activity. When tested on a dataset containing multiple kinds of botnet-based DDoS attack traffic, it demonstrated upwards of 99.8% accuracy, despite a scarcity of training data. When an expanded dataset was collected containing examples of the most commonly used DDoS attack method used by botnets, it achieved perfect accuracy. Additionally, it demonstrates the ability to detect DDoS attack activity within three seconds, even when run on a Raspberry Pi.

The method described in Chapter 5 offers notable improvements over those seen in related works. It retains the high accuracy of deep learning approaches [21, 22, 23] while remaining practical on constrained hardware. As explained in Section 5.3.2, the proposed detection mechanism is well suited for the large network sizes and high traffic volumes characteristic in the IoT. The computational complexity of each component of the system remains constant in relation to network sizes. Similarly, the memory complexity of the data collection module scales linearly with network sizes. These scalability metrics indicate that the method introduced in this thesis eliminates the traditional tradeoff of accuracy for performance which characterizes the field of IoT security.

The research findings presented in this thesis serve as the starting point for several avenues of future work. First, expanding the amount of visual information encoded in a network traffic heatmap could help it reliably identify a greater range of botnet attacks. Currently, the two columns in the network heatmap show the number of packets coming and going from each host. Including more columns by splitting the packet counts by protocol and header flags, could better

expose the patterns of other botnet-related activity, such as port scanning and infection. Second, the application of transfer learning [25] could drastically reduce the amount of data necessary to tailor the detection mechanism to a particular network, reducing the difficulty of deploying the system in new environments.

8 REFERENCES

[1] D. Miorandi, S. Sicari, F. De Pellegrini and I. Chlamtac, "Internet of Things: Vision, Applications and Research Challenges," Ad Hoc Networks, vol. 10, no. 7, 2012, pp. 1497 - 1516.

[2] M. C. Domingo, "An Overview of the Internet of Things for People with Disabilities," Journal of Network and Computer Applications, vol. 35, no. 2, 2012, pp. 584-596.

[3] H. Ghayvat, S. Mukhopadhyay, X. Gui and N. Suryadevara, "WSN and IoT-based Smart Homes and Their Extension to Smart Buildings," Sensors, vol. 15, no. 5, 2015, pp. 10350-10379.

[4] E. Orsi and S. Nesmachnow, "Smart Home Energy Planning Using IoT and the Cloud," in 2017 IEEE UROCON, Montevideo, 2017.

[5] F. Al-Turjman and M. Abujubbeh, "IoT-Enabled Smart Grid via SM: an Overview," Future Generation Computer Systems, vol. 96, 2019, pp. 579-590.

[6] S. Joseph and E. A. Jasmin, "Stream Computing Framework for Outage Detection in Smart Grid," in 2015 International Conference on Power, Instrumentation, Control, and Computing, Thrissur, 2015.

[7] P. V. Dudhe, N. V. Kadam, R. M. Hushangabade and M. S. Deshmukh, "Internet of Things (IoT): An Overview and Its Applications," in 2017 Internation Conference on Energy, Communication, Data Analytics and Soft Computing, Chennai, 2017.

[8] T. Wu, F. Wu, J.-M. Redoute and M. R. Yuce, "An Autonomous Wireless Body Area Network Implementation Towards IoT Connected Healthcare Applications," IEEE Access, vol. 5, 2017, pp. 11413-11422.

[9] C. Dinh-Le, r. Chuang, S. Chokshi and D. Mann, "Wearable Health Technology and Electronic Health Record Integration: Scoping Review and Future Directions," JMIR Mhealth Uhealth, vol. 7, no. 0, 2019.

[10] N. E. Oweis, C. Aracenay, W. George, M. Oweis, H. Soori and V. Snasel, "Internet of Things: Overview, Sources, Applications and Challenges," in Proceedings of the Second International Afro-European Conference for Industrial Advancement AECIA 2015, 2016.

[11] K. Rose, S. Eldridge and L. Chapin, "The Internet of Things: an Overview," The Internet Society (ISOC), vol. 80, 2015, pp. 1-50.

[12] W. Iqbal, H. Abbas, M. Daneshmand, B. Rauf and Y. A. Bangash, "An In-Depth Analysis of IoT Security Requirements, Challenges, and Their Countermeasures via Software-

Defined Security," IEEE Internet of Things Journal, vol. 7, no. 10, 2020, pp. 10250-10276.

[13] S. H. Shah and I. Yaqoob, "A survey: Internet of Things (IoT) Technologies, Applications and Challenges," in IEEE International Conference on Smart Energy Grid Engineering, Oshawa, 2016.

[14] G. Kambourakis, C. Kolias and A. Stavrou, "The Mirai Botnet and the IoT Zombie Armies," in MILCOM 2017 - 2017 IEEE Military Communications Conference (MILCOM), 2017, pp. 267-272.

[15] C. Kolias, G. Kambourakis, A. Stavrou and J. Voas, "DDoS in the IoT: Mirai and Other Botnets," Computer, vol. 50, no. 7, 2017, pp. 80-84.

[16] K. Kalkan, G. Gur and F. Alagoz, "Filtering-Based Defense Mechanisms Against," IEEE Systems Journal, vol. 11, no. 4, 2017, pp. 2761-2773.

[17] X. Liu, X. Yang and Y. Lu, "To Filter or to Authorize: Network-Layer DoS Defense Against Multimillion-Node Botnets," in Proceedings of the ACM SIGCOMM 2008 conference on Data communication, 2008, pp. 195-206.

[18] K. Bhardwaj, J. C. Miranda and A. Gavrilovska, "Towards IoT-DDoD Prevention Using Edge Computing," in USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18), 2018.

[19] A. Wang, W. Chang, S. Chen and A. Mohaisen, "Delving Into Internet DDoS Attacks by Botnets: Characterization and Analysis," IEEE/ACM Transactions on Networking, vol. 26, no. 6, pp. 2843-2855, 2018.

[20] K. Alieyan, M. M. Kadhum, M. Anbar, S. U. Rehman and N. K. Alajmi, "An overview of DDoS attacks based on DNS," in 2016 International Conference on Information and Communication Technology Convergence (ICTC), IEEE, 2016, pp. 276-280.

[21] A. A. Diro and N. Chilamkurti, "Distributed Attack Detection Scheme Using Deep Learning Approach for Internet of Things," Future Generation Computer Systems, vol. 82, 2018, pp. 761-768.

[22] Y. Meidan, M. Bohadana, Y. Mathov, Y. Mirsky, A. Shabtai, D. Breitenbacher and Y. Elovici, "N-BaIoT—Network-Based Detection of IoT Botnet Attacks Using Deep Autoencoders," IEEE Pervasive Computing, vol. 17, no. 3, 2018, pp. 12-22.

[23] C. D. McDermott, F. Majdani and A. V. Petrovski, "Botnet Detection in the Internet of Things using Deep Learning Approaches," in 2018 International Joint Conference on Neural Networks (IJCNN), IEEE, 2018, pp. 1-8.

[24] S. S. Bhunia and M. Gurusamy, "Dynamic Attack Detection and Mitigation in IoT Using SDN," in 2017 27th International Telecommunication Networks and Applications Conference (ITNAC), 2017, pp. 1-6.

[25] I. Goodfellow, Y. Bengio and A. Courville, Deep Learning, MIT Press, 2016.

[26] C. Metz, "Why A.I. and Cryptocurrency Are Making ONe Type of Computer Chip Scarce," the New York Times, 8 May 2018. [Online]. Available: https://www.nytimes.com/2018/05/08/technology/gpu-chip-shortage.html. [Accessed 27 October 2020].

[27] IBM, "NVIDIA GPUs on IBM Cloud servers | IBM," [Online]. Available: https://www.ibm.com/cloud/gpu. [Accessed 27 October 2020].

[28] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto and H. Adam, "Mobilenets: Efficient Convolutional Neural Networks for Mobile Vision Applications," arXiv preprint arXiv:1704.04861, 2017.

[29] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally and K. Keutzer, "SqueezeNet: AlexNet-Level Accuracy With 50x Fewer Parameters and< 0.5 MB Model Size," arXiv preprint arXiv:1602.07360, 2016.

[30] K. Ashton, "That 'Internet of Things' Thing | RFID Journal," 22 June 2009. [Online]. Available: https://www.rfidjournal.com/that-internet-of-things-thing. [Accessed 8 October 2020].

[31] D. Ratasich, F. Khalid, F. Feissler, R. Grosu, M. Shafique and E. Bartocci, "A Roadmap Toward the Resilient Internet of Things for Cyber-physical Systems," IEEE Access, vol. 7, 2019, pp. 13260-13283.

[32] L. Da Xu, W. He and S. Li, "Internet of Things in Industries: A Survey," IEEE Transactions on Industrial Informatics , vol. 10, no. 4, 2014, pp. 2233-2243.

[33] Google, "Google Home- Apps on Google Play," [Online]. Available: https://play.google.com/store/apps/details?id=com.google.android.apps.chromecast.app&hl=en_US&gl=US. [Accessed 10 October 2020].

[34] Amazon, "Amazon Alexa Official Site: What is Alexa?," [Online]. Available: https://developer.amazon.com/en-US/alexa. [Accessed 9 October 2020].

[35] C. De Canniere, O. Dunkelman and M. Knevzevic, "KATAN and KTANTAN- a Family of Small and Efficient Hardware-Oriented Block Ciphers," in International Workshop on Cryptographic Hardware and Embedded Systems, Springer, 2009, pp. 272-288.

[36] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks and L. Wingers, "SIMON and SPECK: Block Ciphers for the Internet of Things," IACR Cryptol. ePrint Arch., vol. 2015, 2015, p. 585.

[37] R. Mahmoud, T. Yousuf, F. Aloul and I. Zualkernan, "Internet of Things (IoT) Security: Current Status, Challenges and Prospective Measures," in 2015 10th International Conference for Internet Technology and Secured Transactions (ICITST), IEEE, 2015, pp. 336-341.

[38] M. D. Alshehri and F. K. Hussain, "A Centralized Trust Management Mechanism for the Internet of Things (CTM-IoT)," in International Conference on Broadband and Wireless Computing, Communication and Applications, Springer, 2017, pp. 533-543.

[39] F. Mwagwabi, T. McGill and M. Dixon, "Short-Term and Long-Term Effects of Fear Appeals in Improving Compliance with Password Guidelines," Communications of the Association for Information Systems, vol. 42, no. 1, 2018, p. 7.

[40] F. Merces, "Miner Malware Targets IoT, Offered in the Underground," trendmicro, 2 May 2018. [Online]. Available: https://www.trendmicro.com/en_us/research/18/e/cryptocurrency-mining-malware-targeting-iot-being-offered-in-the-underground.html. [Accessed 10 October 2020].

[41] K. A. Fakeeh, "An Overview of DDOS Attacks Detection and Prevention," International Journal of Applied Information Systems (IJAIS), vol. 11, no. 7, 2016, pp. 25-34.

[42] G. Somani, M. S. Gaur, D. Sanghi, M. Conti, M. Rajarajan and R. Buyya, "Combating DDoS Attacks in the Cloud: Requirements, Trends, and Future Directions," IEEE Cloud Computing, vol. 4, no. 1, 2017, pp. 22-32.

[43] P. Kamboj, M. C. Trivedi, V. K. Yadav and V. K. Singh, "Detection Techniques of DDoS Attacks: a Survey," in 2017 4th IEEE Uttar Pradesh Section International Conference on Electrical, Computer and Electronics (UPCON), IEEE, 2017, pp. 675-679.

[44] H. A. Herrera, W. R. Rivas and S. Kumar, "Evaluation of Internet Connectivity Under Distributed Denial of Service Attacks from Botnets of Varying Magnitudes," in 2018 1st International Conference on Data Intelligence and Security (ICDIS), IEEE, 2018, pp. 123-126.

[45] S. L. Feibish, Y. Afek, A. Bremler-Barr, E. Cohen and M. Shagam, "Mitigating DNS Random Subdomain DDoS Attacks by Distinct Heavy Hitters Sketches," in Proceedings of the fifth ACM/IEEE Workshop on Hot Topics in Web Systems and Technologies, 2017, pp. 1-6.

[46] B. Krebs, "New Mirai Worm Knocks 900K Germans Offline - Krebs on Security," 30 November 2016. [Online]. Available: https://krebsonsecurity.com/2016/11/new-mirai-

worm-knocks-900k-germans-offline/. [Accessed 21 October 2020].

[47] R. Hallman, J. Bryan, G. Palavicini, J. Divita and J. Romero-Mariona, "IoDDoS- the Internet of Distributed Denial of Service Attacks," in 2nd international conference on internet of things, big data and security. SCITEPRESS, 2017, pp. 47-58.

[48] J. Gamblin, "jgamblin/Mirai-Source-Code: Leaked Mirai Source Code for Research/IoC Development Purposes," [Online]. Available: https://github.com/jgamblin/Mirai-Source-Code. [Accessed 21 October 2020].

[49] GNU, "Wget- GNU Project- Free Software Foundation," [Online]. Available: https://www.gnu.org/software/wget/. [Accessed 21 October 2020].

[50] K. Sollins, "RFC 1350- The TFTP Protocol (Revision 2)," July 1992. [Online]. Available: https://tools.ietf.org/html/rfc1350. [Accessed 21 October 2020].

[51] CAPTCHA, "The Official Captcha Site," [Online]. Available: http://www.captcha.net/. [Accessed 22 October 2020].

[52] H. Om and A. Kundu, "A Hybrid System for Reducing the False Alarm Rate of Anomaly Intrusion Detection System," in 2012 1st International Conference on Recent Advances in Information Technology (RAIT), IEEE, 2012, pp. 131-136.

[53] X. Yuan, C. Li and X. Li, "DeepDefense: Identifying DDoS Attack Via Deep Learning," in 2017 IEEE International Conference on Smart Computing (SMARTCOMP), IEEE, 2017, pp. 1-8.

[54] J. Evangelho, "What's Really Going On With Nvidia RTX 30 Series Supply?," Forbes, 6 October 2020. [Online]. Available: https://www.forbes.com/sites/jasonevangelho/2020/10/06/nvidia-ceo-rtx-30-series-has-a-demand-problem-not-a-supply-problem/#75d536723277. [Accessed 27 October 2020].

[55] N. Koroniotis, N. Moustafa, E. Sitnikova and B. Turnbull, "Towards the Development of Realistic Botnet Dataset in the Internet of Things for Network Forensic Analytics: Bot-IoT Dataset," Future Generation Computer Systems, vol. 100, 2019, pp. 779-796.

[56] The pandas development team, pandas-dev/pandas: Pandas, Zenodo, 2020.

[57] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith and others, "Array Programming with NumPy," Nature, vol. 585, no. 7825, 2020, pp. 357-362.

[58] M. Waskom and t. S. D. Team, "mwaskom/seaborn," Zenodo, September 2020. [Online]. Available: https://doi.org/10.5281/zenodo.592845.

[59] J. D. Hunter, "Matplotlib: A 2D Graphics Environment," Computing in Science & Engineering, vol. 9, no. 3, 2007, pp. 90-95.

[60] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang and Z. DeVito, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in Advances in Neural Information Processing Systems 32, Curran Associates, Inc., 2019, pp. 8024-8035.

[61] "MobileNet v2 | PyTorch," [Online]. Available: https://pytorch.org/hub/pytorch_vision_mobilenet_v2/. [Accessed 1 November 2020].

[62] "SqueezeNet | PyTorch," [Online]. Available: https://pytorch.org/hub/pytorch_vision_squeezenet/. [Accessed 1 November 2020].

[63] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and E. Duchesnay, "Scikit-Learn: Machine Learning in Python," Journal of Machine Learning Research, vol. 12, 2011, pp. 2825-2830.

[64] Raspberry Pi Foundation, "Raspberry Pi 4 Model B specificaitons - Raspberry Pi," [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/specifications/?resellerType=home. [Accessed 1 November 2020].

[65] Intel, "Intel Xeon Processor E5-2440 (15M Cache, 2.40 GHz, 7.20 GT/s Intel® QPI) product specifications," [Online]. Available: https://ark.intel.com/content/www/us/en/ark/products/64612/intel-xeon-processor-e5-2440-15m-cache-2-40-ghz-7-20-gt-s-intel-qpi.html. [Accessed 2 November 2020].

[66] T. Adegbija, A. Rogacs, C. Patel and A. Gordon-Ross, "Micro-Processor Optimizations for the Internet of Things," arXiv preprint arXiv: 1603.02393, 2016.

[67] D. Merkel, "Docker: Lightweight Linux Containers for Consistent Development and Deployment," Linux Journal, vol. 2014, no. 239, p. 2.

[68] S. Klabnik and C. Nichols, The Rust Programming Language, No Starch Press, 2018.

[69] Nmap, "Nping - Network packet generation tool / ping utility," [Online]. Available: https://nmap.org/nping/. [Accessed 2 November 2020].

[70] "D.2. ip route," [Online]. Available: http://linux-ip.net/html/tools-ip-route.html. [Accessed 2 November 2020].

[71] "iptables(8) - Linux man page," [Online]. Available: https://linux.die.net/man/8/iptables. [Accessed 30 October 2020].

[72] S. Benitez, "Rocket - Simple, Fast, Type-Safe Web Framework for Rust," [Online]. Available: https://rocket.rs/. [Accessed 2 November 2020].

[73] M. Grinber, Flask Web Development: Developing Web Applications with Python, O'Reilly Media Inc., 2018.

APPENDIX

Docker was utilized to provide virtual hosts and networking for the smart home simulation illustrated in Figure 11. The entire simulation is comprised of three networks: the smart home access network, an analysis network, and the Internet. The smart home access network consisted of simulated IoTDs and an access router. The analysis network contains the data formatting module, the data analysis module, and a database used to enable persistent data storage between simulation scenarios. Hosts not included in the botnet attack detection system and smart home simulation were in the Internet. For the purposes of the experiments described in Chapter 6, these hosts included the botnet CnC server and a victim host for the bots to attack.

Smart Home Simulation

Simulated IoTDs. The smart home network contains the seven IoTDs described in Table 8. The IoTDs are either publishers, subscribers, or both. Publishers generate randomized data describing their state and send it via a TCP socket client to each device subscribing to them. Subscribers host a TCP socket server which accepts incoming connections from the devices they publish to. In the case that an IoTD is both a publisher and a subscriber, the processes for these roles are run in separate threads. Each IoTD regardless of their designation hosts a background TCP server listening for attack commands from the CnC server. This attack command contains the IP address of the target and the duration the attack should last. Attacks are carried out by starting a child process running the Nping packet generation tool in TCP flood mode.

IoT Access Router. An eighth host on the IoT access network serves as the network's router. On startup, each simulated IoTD invokes the ip route command to funnel all their traffic

through the router. Iptables is used to establish counting for the ingressing and egressing packets for each IoTD on the simulated network. The router hosts an HTTP endpoint which, upon receiving a get request from the data formatting module, retrieves and resets the packet counts then replies with them in JSON format.

Analysis Network

Database. Docker allows containers running database instances to access the host computer's file system, enabling persistent data storage between simulation sessions. A Mongo database is utilized in this role. As illustrated in Listing A-1, each entry in the database corresponds to a single network traffic window. Aside from the packet counts for each host on the network, the length and classification of the traffic windows are stored for each entry in the database.

```
{
     "_id": {"$oid": "5f74ee44a482c6e9903b38cc"},
     "smart_home_controller": {"incoming": 61, "outgoing": 55},
     "weather_sensor": {"incoming": 60, "outgoing": 64},
     "thermostat": {"incoming": 31, "outgoing": 30},
     "garage_door": {"incoming": 0, "outgoing": 0},
     "refrigerator": {"incoming": 45639, "outgoing": 45925},
     "lights": {"incoming": 0, "outgoing": 0},
     "motion_sensor": {"incoming": 0, "outgoing": 0},
     "label": "ddos_tcp"
}
```

Listing A-1: An example entry from the database

Data Formatting Module. The data formatting module receives the current network traffic window length and operational mode for the network simulation on startup. At the end of each network traffic window, it sends an HTTP get request to the IoT access router, which

responds with the packet counts for each IoTD on the network in JSON format. If the network is currently in interactive mode, the data formatting module uses these packet counts to create a network traffic heatmap, which it sends to the data analysis module via HTTP post request. If the network is in data collection mode, the data formatting module is also informed whether attack or normal data is currently being collected. It stores the JSON packet counts for each traffic window in a Mongo DB database along with its corresponding label and traffic window length. In testing mode, it queries the database for all entries corresponding with the current network traffic window length. It then iterates over each network traffic window, creates a heatmap from it, and sends this heatmap to the traffic analysis module via HTTP post request.

Data Analysis Module. On startup, the data analysis module loads either SqueezeNet or MobileNet to use for classification. It hosts an HTTP endpoint, which accepts post requests from the data formatting module. When a request is received, it runs the current CNN on the heatmap it contains and returns its classification in response.

The Internet

Botnet CnC Server. The CnC server is only run in interactive and data collection mode. In interactive mode, it hosts a web interface which allows a user to view the current IoTDs on the smart home access network, select any combination of them, and instruct them to begin an attack lasting a specified period of time. In data collection mode, it runs a continuous script which randomly selects a set of IoTDs to participate in an attack. It varies the number of attackers iteratively from one to seven, ensuring each possibility receives roughly even representation in the resulting dataset.

DDoS Target. The DDoS victim is a docker container running a continuous bash

process. This provides a valid host which other Docker containers can establish a network connection with. It responds to incoming TCP connections with an error message stating that no process is waiting for connections at the destination port.