Botnet Network Intrusion Detection on Resource Constrained Devices for IoT Applications

Patrick Shortall
Department of Computer Science
Dublin City University
Dublin, Ireland
patrick.shortall2@mail.dcu.ie

Dr. Darragh O'Brien
Department of Computer Science
Dublin City University
Dublin, Ireland
darragh.obrien@dcu.ie

***Abstract*—The Internet of Things (or IoT) is widely becoming more integrated into society, from smart household devices to real-time vehicle diagnostics. However, many of these newly internet-enabled devices are fundamentally insecure, leaving them open to an array of attacks, most prominent of which are botnets. As shown with the Mirai botnet attack, IoT devices can be exploited for use in large-scale Distributed Denial of Service attacks. In this paper, a number of lightweight classification algorithms are evaluated for use as part of a Network Intrusion Detection System to identify botnet attacks. A simulation of the system is implemented and tested for performance overhead on a Raspberry Pi 3 single-board computer. Ultimately it is found that the Bonsai tree classifier and XGBoost perform best in terms of accuracy vs model size.**

Keywords—Network Intrusion Detection, Internet of Things, botnets, Raspberry Pi

I. Introduction

Physical devices and everyday objects with built-in internet connectivity, or Internet of Things (IoT) devices are rapidly becoming more of a part of our daily lives. These devices are not limited to any specific industry, with their use growing in a variety of areas such as consumer electronics and appliances, city infrastructure including smart grid systems and e-health, with network enabled devices for patient monitoring and drug delivery [1]. The growing role of these network-enabled devices in physical systems means that ensuring that they are not compromised is vitally important to the integrity of their application.

In the past, the vast and homogeneous nature of IoT devices has made them ideal targets for use in botnet attacks [2]. One of the most notable of these was the Mirai botnet which in September 2016 caused the website of security journalist Brian Krebs and French Web host OVH to be taken offline, and was later responsible for the October 2016 Dyn (a major DNS provider) cyberattack [3]. Due to the widespread use of default credentials or out-of-date firmware on IoT devices, these devices have proven to be ideal targets for botnet infection. Previous reports have shown that the primary purpose of IoT malware deployment as recently as September 2018, is still to perpetrate DDoS attacks through botnets [4]. For these reasons, it is evident why ensuring the security of IoT devices is of the utmost importance. However, unlike conventional computing systems, Internet of Things devices are characterised by their constraint nature. They are often battery operated, with only a small amount of computational power at their disposal. These restrictions mean that implementing existing security measures to prevent the spread of botnets and other attacks on IoT devices is not always possible. As these devices become steadily integrated into more aspects of our lives the necessity to create proper security measures for them, that cater to their restrictions is becoming an ever-present need.

The typical architecture of an IoT system can be reduced to three main layers: the perception layer (where sensors and actuators, or edge devices collect and send on data), the network layer (where data is aggregated and potentially processed on a gateway device) and the application layer (generally a cloud-based server which performs a task on the data or triggers an action due a change in data) [5]. Within this broad framework, a network gateway device has the potential to act as a very effective point to assess the security of the edge nodes connected to it. This is because the edge devices are the most constrained part of the architecture, generally with just enough computational power to carry out the sensing or actuating task they are designed for.

The application layer offers a wealth of cloud-based resources and capabilities, but processing at this level introduces latency and hinders the ability of the system to detect threats in real-time. This is why the gateway, with its position between these two layers makes it an ideal device to monitor the activity of the edge nodes, using a network intrusion detection system (NIDS).

Although gateways in IoT systems do not suffer the same constraints as edge nodes, their resources are still limited. Typically, these devices may have similar specifications to that of a single-board computer [6]. The limited RAM and processing power on such devices mean that the design of a detection framework must be lightweight enough not to affect the gateway's other functionality such as the aforementioned data aggregation and protocol conversion.

Within conventional network intrusion detection systems there are generally two main approaches: signature-based detection and anomaly detection. Although signature-based systems have more commonly been used in industry due to their low false positive rate, the rapid advancements in the field of machine learning (ML) have made the possibility of anomaly detection-based systems more of a reality. One of the main advantages of anomaly detection is its ability to detect previously unseen attacks, which makes it an ideal candidate for use in the ever changing landscape of botnet-based threats.

Conventionally, the words anomaly detection have signified the use of unsupervised machine learning models such as K-Means Clustering and Local Outlier Factor or semi-supervised methods like One-Class Support Vector Machines. However, in more recent times the term has also been applied to supervised learning classification in the context of network intrusion detection systems [7] [8]. With a wealth of research being conducted on supervised learning in recent years, it is possible to find a number of models that fit the constrained

nature of IoT gateway deployment, for use as part of a network intrusion detection system. In this paper, five lightweight machine learning models are evaluated for this purpose and are deployed as part of a simulated NIDS in a framework designed for this application on a Raspberry Pi 3 single board computer [9], a device with similar specifications to an average constrained IoT gateway.

The rest of this paper is laid out as follows: in section II previously conducted related research is examined and discussed in the context of this project, in section III the five classification algorithms are outlined, the features of the dataset used are evaluated and the design of the simulated framework is detailed, in section IV the metrics used and the performance of both the classification algorithms on the application and the simulated framework are tested and evaluated, and finally section V concludes the paper.

II. Background & Related Work

A. Intrusion Detection in IoT

Intrusion detections can be defined as any malicious intrusion or attack on network vulnerabilities, computers or information systems, that may give rise to serious disasters and violate one or more of the three pillars of security: confidentiality, integrity and availability [10]. Intrusion detection systems can for the most part, be broken down into two main categories: host-based and network-based frameworks. Host-based systems, as the name would suggest are implemented on individual hosts, inspecting system events on the device. With IoT networks being characterised by their vast number of constrained edge nodes, this is clearly an impractical solution in this space. Network-based systems on the other hand, typically inspect network events at a single collecting point in the network. They are used to monitor and analyse network traffic and protect the system against potential threats.

As previously mentioned, the method by which the traffic is analysed can be further split down into misuse or signature-based detection and anomaly detection. Within the field of anomaly detection, a plethora of research has been conducted using machine learning methods. As shown in [11], there are many different possibilities of using unsupervised and supervised approaches for network intrusion detection, depending on the application needed. Over the last two decades, these machine learning-based systems have come to rival their signature-based counterparts [12].

Anomaly detection is best suited to this project due to the nature of IoT traffic. While laptops and smartphones may access a large range of web endpoints as a result of web browsing activity, IoT devices are generally task-oriented, such as an edge node sending temperature values to a fixed end point consistently over an allotted time window. This means that IoT network activity is generally more predictable and structured, with less complex network protocols and variance than PCs. Therefore, it is clear that using flow level traffic statistics (how packets are sent) for generating feature data instead of deep packet inspection (what is in a packet) is a viable option in this domain.

As detailed in [13], lightweight features are needed for detection systems to scale to high bandwidth traffic and large networks. Furthermore, along with being a practical solution for IoT systems, the use of lightweight flow-based features requires considerably less processing costs than its packet inspection-based counterparts [14]. Finally, the use of transactional flow statistics also ensure that the NID system can create protocol agnostic features, giving it the capabilities to capture a variety of different protocols.

B. Botnet Detection in IoT

In terms of the range of threats posed against Internet of Things devices, none have had as large an impact as botnets [3]. Botnets can be described as a number of connected computing devices that are controlled remotely by a command and control server, to carry out malicious activity, most notable of which are Distributed Denial of Service attacks [15]. As previously mentioned, the scale and severity of the Mirai botnet attacks were unparalleled, with the French web host and cloud service provider OVH being hit by a DDoS attack that peaked at 1.1 Tbps, and the second Mirai attack later that year against the Dyn service provider which caused hundreds of websites including Twitter, Netflix and GitHub to be taken offline temporarily (with an attack peaking at 1.2 Tbsp.) [16]. Infecting IoT devices as part of botnets is clearly one of the greatest threats to IoT devices at the present time, and as these botnets have only gotten more sophisticated since the 2016 Mirai attacks, the need for a suitable system to detect when these devices are compromised is a necessity.

Of the work previously conducted in the area of IoT botnet detection, a number of approaches have been taken. In [17], a network-based anomaly detection framework was created using deep autoencoders to detect compromised IoT devices. Like many papers in this area, a dataset was created for the research, with both Mirai and Bashlite (another significant botnet) used to create attack traffic. The features generated for the purposes of training the autoencoders were a number of flow-based statistics captured over a group of time windows ranging up to one minute in size. The autoencoders classified data based on one of three classes of benign, Bashlite or Mirai. Ultimately the paper found that the autoencoders performed extremely well, with a false positive rate of zero on most of the nine IoT devices used as edge nodes in their dataset. However due to the scale of the model, deployment on the application layer would be the only viable option. This, coupled with the very large time windows used, make the model less suitable for real-time use.

Similarly in [18], a detection model was created from a Recurrent Neural Network based on Bidirectional Long Short Term Memory (BLSTM-RNN). This model uses packet-based features, created by using word embedding for text recognition and converting attack packets into a tokenized integer format. Five classes of data were used in this paper for classifying the attack vectors. These included normal, Mirai (botnet scanning for new device), udp, dns and ack (the last three of which were all simulated flooding attacks). The paper ultimately found that this model performed extremely well on the dataset, however this framework is still very much reliant on the use of a large quantity of computing resources, both for the feature extraction and classification. A more lightweight deep learning-based approach was taken in [19], where Su et al. propose the use of a Convolutional Neural Network to classify DDoS malware families in IoT. This was achieved by extracting one-channel grey-scale images from binaries on the devices. This system negated the constraints of IoT devices by proposing a two-tier detection architecture based on local detection and cloud-based classification. In this framework, suspicious activity was sent on to the server-based classifier for further analysis. Three classes of benign, Gafgyt (another

prominent botnet) and Mirai were used for training purposes. This scheme ultimately produced accuracy results as high as 94% and time consumption values for the classifier in the low millisecond range.

A very thorough evaluation of datasets that were created or used for botnet detection was done in [20]. This paper found that most datasets used as part of research in this area suffered from a lack of inclusion of IoT traffic, were not realistic or did not include realistic attack scenarios. A new dataset was then proposed and created to address these issues, with simulated attack traffic created to represent information gathering, denial of service and information theft scenarios in botnet attacks. The dataset created in this paper was chosen for training the classifiers in this project because of this considered approach. The paper also evaluated the use of three classifiers on the newly created dataset: a Support Vector Machine (SVM), a Recurrent Neural Network (RNN) and a Long Short Term Memory RNN. Each of the three classifiers were trained and tested on binary class labels of normal and attack, with the SVM ultimately performing best.

One of the papers evaluated in [20], that was found to use a questionably method for including attack traffic was [7]. Although this aspect of the paper was questioned, the paper also raised a number of very astute points about the placement of a NIDS in the IoT architecture. Roshi et al. reasoned that a network gateway is in a unique position with the ability to "inspect, store, manipulate, and block any network traffic that crosses its path". Although the classification algorithms were not implemented on it, a raspberry Pi model 3 was used to simulate the gateway during the dataset creation. Finally, the paper also stipulated that the constrained nature of the gateway must be factored into the NIDS' design, with a low memory implementation, stating that an optimal algorithm should be stateless or at the very least, require storing flow information over short windows of time only. With this in mind, previous research must also be evaluated in terms of how lightweight a systems is.

C. *Performance Overhead in Network Intrusion Detection Systems*

A range of surveys have been conducted into NIDS in the field of IoT, such as [21], however many of the systems evaluated in these papers fail to examine the performance and instead focus more on traditional metrics such as accuracy. However in [22], existing systems in this area were analysed in terms of computational overhead, energy consumption and privacy implications. This paper ultimately found that many accuracy-optimised frameworks had been created that neglected the issues of limited size and processing power of IoT devices. Nevertheless, a number of papers had measured performance overhead using metrics such as CPU usage, latency, RAM usage and energy consumption. This paper also found that the Raspberry Pi was used as a viable device to simulate an intrusion detection system in a number of papers, however all of these systems used signature-based detection frameworks.

D. *Lightweight Classfication*

One important aspect of creating a more lightweight NIDS for gateway deployment, is the choice of classification algorithm to implement. The five classification algorithms utilized in this project are discussed in section III. Two other novel approaches that were tested are outlined below. An interesting approach to reducing model size was taken in [23]. In this paper, existing machine learning models and their respective datasets were ported to constraint embedded microcontrollers through the use of source code generation. This allowed these algorithms to work with high accuracy and model sizes as low as 2,500 Bytes.

A number of papers have been written about research conducted on constraining the size of deep learning models, however one of the most notable has been the work by a Google research team into Morphnet [24]. Morphnet is a highly developed technique for refining neural network models. It allows optimisation of existing models under a range of constraints, including FLOPS (floating point operations per second) count, latency and model size. This is done through a cycle of shrinking and expanding phases to find and remove inefficient neurons. The research was applied to large-scale image processing models, ultimately reporting a large reduction in model size and FLOPS count, with very little loss in quality.

Overall, from all of the research discussed in this section, it is clear that network intrusion detection systems have been created previously for detecting compromised IoT devices by botnets. However, many of these systems have trained models using binary normal and attack classes such as [7] and [20] or models that only distinguish between benign behaviour and two different botnet types [17] [19]. Far fewer such as [18], have trained machine learning classifiers to identify more specific types of botnet network traffic behaviour and train models based on this. Similarly, many of these systems have not been evaluated consistently in terms of the constraints of IoT deployment.

III. Methodology & Implementation

To begin, an overview of each of the five lightweight classification algorithms will be given before the feature selection process and the design and implementation of the overall system is outlined.

A. *ProtoNN*

ProtoNN, as its name would suggest is a prototype based k-nearest neighbour (kNN) machine learning algorithm, created to address the problem of real-time and accurate prediction on resource-scarce devices in the IoT domain [25]. The algorithm's creators stipulated that traditionally kNN suffers from three main issues in this application setting: poor accuracy due to a lack of clarity about the choice of distance metric to use, large model size due to kNN requiring the entire training data for prediction and long prediction time since kNN requires that the distance must be calculated between a given test point and each training point.

The algorithm tries to solve these problems using three main ideas: firstly, a small number of prototypes are learned to represent the entire training set. This allows for a smaller model size and the flexibility for the algorithm to generalise to multi-class problems, as used in this project. Secondly, the algorithm uses sparse low-dimensional projection to project the full data using a sparse projection matrix. This provides much better accuracy in the projected space and ensures that the full training data doesn't have to be saved for prediction, meaning the model size is reduced. Finally, by joint discriminative learning of the projection and prototypes with an explicit model size constraint, ProtoNN is able to obtain an optimal model within a given model size without the need for post-pruning to force the model to fit in memory.

The algorithm was tested on six different binary classification datasets and five multi-class datasets, with six other algorithms tested on all eleven datasets for comparison. It was found that ProtoNN achieved high accuracy results across both binary and multi-class problems, with a much smaller model size than the other algorithms used for comparison.

In terms of the implementation of ProtoNN used in this project, source code for the algorithm using python and TensorFlow was taken from [26]. After testing, an optimal learning rate of 0.05 was found. The number of epochs were reduced to 20, to allow for a similar training time to the other algorithms. A batch size of 32 was also used, as specified in the algorithm's original implementation. It was determined that a larger batch size was too high use on the Raspberry Pi.

B. Bonsai

Bonsai is a strong and shallow non-linear tree based classifier [27], designed for implementation in extremely resource-scarce applications. The algorithm maintains accuracy while also minimising model size and prediction cost using three main design considerations: first, the tree-based model learns a single, shallow and sparse tree with robust nodes for accurate prediction. Second, internal and leaf nodes in Bonsai make non-linear predictions. The overall prediction for a point is the sum of the individual node predictions along the path encountered by the point. This allows the model to learn non-linear decision boundaries, something that traditional decision trees often encounter problems with. This also allows for sharing of parameters along paths to occur, which reduces model size and allows for more accurate results when dealing with multi-class problems.

Much like ProtoNN, Bonsai also sparsely projects all data into a low-dimensional space, in this case, in which the tree is learnt, again allowing for smaller model size. Also similar to ProtoNN, in Bonsai all nodes are jointly learned along with the sparse projection matrix. Doing this instead of greedily learning the tree node by node facilitates the optimal allocation of memory to each node while also maximising prediction accuracy.

Bonsai was tested on seven different binary classification datasets of varying types, against four other machine learning models: gradient boosting decision tree, K-nearest neighbour, a support vector machine and a small neural network. Bonsai proved to perform as well, and in some cases better than the other models, with a much smaller model size. The algorithm performed similarly when evaluated on three different multi-class classification datasets against the same four algorithms.

The source code for the implementation of Bonsai was taken from [28]. A learning rate of 0.01 was found to produce optimal results without increasing model size. Again, just 20 epochs were used, to reduce training time (it was also found that minimal improvements were made using more). A tree depth of 3 was used to maintain a small model size.

C. Super Fast Support Vector Classifier

The Super-Fast Support Vector Classifier [29] is an optimized implementation of the algorithm created by the same research group in [30]. It consists of a single layer neural network with a radial-basis hidden layer. The hidden layer contains units with the same radius and an output that produces only binary weights. Part of the speed-efficiency of the algorithm stems from the linear number of hidden neurons used. Similar to standard Support Vector Machines (SVMs), the centre of these neurons are not learned but selected as 'support vectors' from among the samples of the training data. However, unlike SVMs a very simple selection algorithm is used in SFSVC based on a novel detection principle using the size of the overlapping factor.

In terms of the implementation created in [29], two changes to the algorithms previous architecture were made for improvements in terms of both speed and accuracy. In the previous version, RBF (Radial Basis Function) units were created from browsing the entirety of the training data, however the SFSVC uses a search process for each of the M classes, creating a specific set of RBF units for all units coming from a particular class. This change means a more accurate and faster supervised search is used instead of the previously used unsupervised search. Secondly by implementing distance calculations using matrix multiplication, the algorithm performs up to forty times faster and also accelerates RBF computations on some platforms. Ultimately, the algorithm's lack of extra hidden layers and inclusion of only one tuneable parameter, allow for its relatively small model size.

SFSVC was tested on both the USPS and MNIST handwritten multi-class classification datasets and showed accuracy values over 95% in both cases, with much smaller training and test times than the previous implementations of the algorithm. The implementation used in this project uses the source code created by the researchers who implemented it [31]. The main tuneable parameter of SFSVC: the radius size, was set to 25, following the optimization specifications outlined by the algorithm's authors.

D. XGBoost

As Described by in its documentation, "XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable" [32]. XGBoost (or eXtreme Gradient Boosting) uses an implementation of gradient boosting decision trees that can create models that are highly optimised for both speed and performance. In the context of this application, XGBoost was chosen to be a benchmark to test the smaller classification models against. However, in its own right this algorithm possesses characteristics that fit the needs of this application. Firstly, with most of its needed memory construction being done in the first pass of the algorithm, XGBoost ensures a lack of dynamic memory in its implementation.

Along with this memory optimisation, XGBoost's considered choice of block size prevents issues with stalling in the CPU cache, allowing the algorithm to be more hardware-friendly. The algorithm is also designed to be ideal for scaling to larger datasets while using limited resources, which fit the criteria of the lightweight classification algorithms for this paper. In terms of implementation used in this project, a readily available library was created by its authors [32]. No major changes were made to the basic XGBoost implementation, beyond specifying its use for multi-class classification that uses the softprob objective. This criteria simply specifies that the outputs contain a matrix of probability values for each class to predict.

E. Random Forest

Random Forest was chosen as the fifth classification algorithm, to act as a reference to show a unconstrainted

implementation of supervised machine learning. It was also determined to be a good comparison to the other tree-based classifiers such as Bonsai and XGboost. Random Forest is an ensemble learning algorithm, that consists of a number of decision trees. A basic implementaion was created in this project using the Sci-kit learn library [33], with 100 trees specified in the arguments of the model.

F. Dataset Evaluation

As previously mentioned, the dataset used for training the five classifiers was created in [20]. A number of IoT devices were simulated during the creation of the dataset including: a weather station, smart fridges, smart garage door, motion activated light and a smart thermostat. The network behaviour of these devices was simulated using the Node Red Tool, with javascript code created to mimic each of the devices.

Botnet attacks were deployed using Kali Linux virtual machines to attack each of the simulated IoT devices. The ten attack classes created, were divided up into three groups: two data theft classes (keylogging and data exfiltration), two data gathering classes (service scanning and OS fingerprinting) and six denial of service classes (TCP, UDP and HTTP for both DoS and DDoS). These ten classes show the varying depth of botnet network activity.

Initially all of the models were trained using a ready-made 5% sample of the dataset provided online by the research group [34]. However, after it was decided that a more detailed, multi-class classification model would be more appropriate, the breakdown of the eleven classes in the dataset was evaluated more closely. Upon examining this, it was shown that classes such as Data Exfiltration had as little as two samples in the whole 5% of the dataset used. This would obviously cause inaccurate prediction as a whole, with only four of the eleven classes dominating the currently used dataset.

To remedy this, the 15GB entire dataset was used to create a sampled dataset that more evenly represented the full spectrum of classes for prediction. However, despite these efforts, the two data theft classes: Keylogging and Data Exfiltration were still vastly under-represented, with the latter possessing only 116 samples. For this reason, the two Data Theft classes were removed from the dataset, leaving the dataset with a class distribution illustrated in figure 1 below.

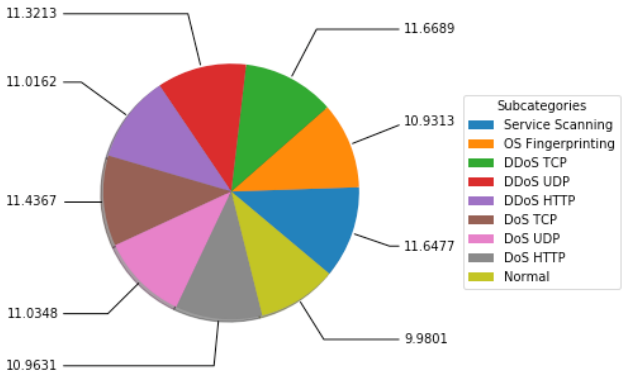

Figure 1: Class Distribution

Despite the lack of the two data theft classes, this multi-class dataset facilitated the ability to train each of the models for more detailed botnet detection. Potentially allowing for the multi-class predictions to be used in real-time to take more specific action against the type of threat detected.

As shown in section II, the features used by a network intrusion detection system in this setting should be lightweight. For this reason, the feature extraction method used to create features in this dataset was replicated using the same Argus tool [35] used in the original dataset generation. For each of the original twenty one features included in this dataset, measurements were taken in terms of the CPU utilisation, latency and maximum resident set size (an approximate value for the required RAM of a process) needed to extract each feature. The perf performance analysis tool was used for measuring the CPU utilisation, the time command was used for measuring latency and the gnu time (usr/bin/time) command was used to measure the maximum resident set size.

Lower values in each of these three metrics indicated that a feature was more lightweight, therefore to simply the feature selection process, the three metrics were aggregated into one resource use value for each feature. This allowed the features to be evaluated more easily against a more traditional statistical measure, in this case: mutual information. Mutual information is a measure of mutual dependence between two discrete variables, used here as a measure between each of the features and the class labels (the target variable). The Sci-kit Learn 'mutual_info_classif()' function was used to implement this. With two distinct metrics to measure feature performance, the values from both were scaled to a range of (-1, 1) and plotted, as shown in Figure 2. From this, a number of features were picked. It was eventually found that the models performed optimally using eight of these features (over the use of six or ten).

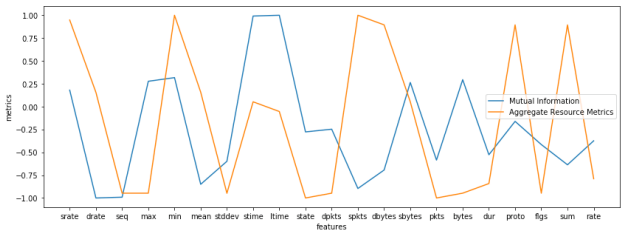

Figure 2: Feature Evaluation

G. System Design & Implementation

As detailed in section II, the overall network intrusion detection system would be placed on a gateway node in the IoT architecture. This is illustrated in Figure 3 below. This would allow for the screening of network traffic as it is aggregated or its protocols are converted, before being relayed to the application layer. Its position in the network would also allow for more real-time response to threats.

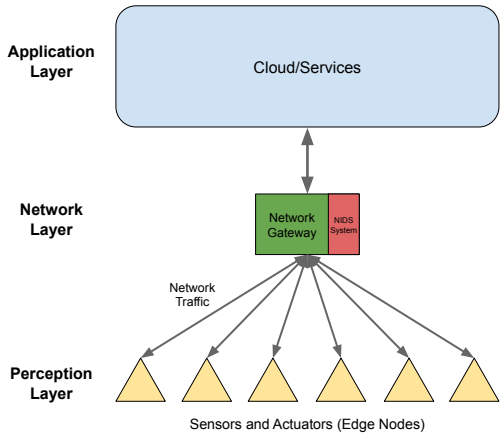

Figure 3: NIDS integration into IoT Architecture

To simulate the real-life conditions of a network gateway, each of the five classifiers were implemented on a Raspberry Pi 3 Model B single board computer. However, purely simulating just the intrusion detection classification process would not be adequate to determine the feasibility of the Raspberry Pi in this application. Therefore, a fully-fledged network capture, feature extraction and classification system was designed and implemented. A single iteration is shown in figure 4. First, the traffic is captured for a specified time-window, dependant on the needs of the network. This window of traffic is then converted into the Argus file format and the eight features are extracted using the ra command. Due to the rapid turnover of traffic, a file removal function was created to remove old traffic after a certain expiry time-window, potentially once it had been declared benign. Text-based features such as protocol and state are then encoded into numerical features, for the classification purposes. Finally, the traffic is classified into one of the nine traffic classes and the process continues.

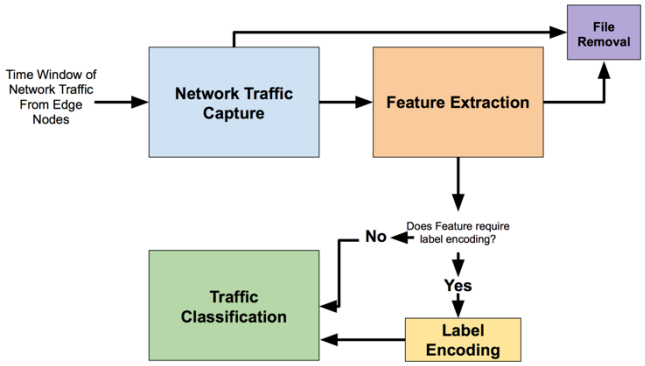

Figure 4: Overall System Design

Initially, all of this system was implemented in Python, however after some resource utilisation testing, a more lightweight version of the system was created, with bash scripts executing the network capture, feature extraction phase and the file removal. For the network capture process, both tshark (a command-line version of Wireshark) and tcpdump were evaluated as outlined in the next section.

IV. Evaluation and Results

A. Metrics

To test the performance of the classification algorithms and the traffic capture process, a number of metrics were used. Some of these metrics, such as accuracy, model size and training time were specific to the five classifiers, however a number of resource metrics such as average CPU utilisation, average memory utilisation and energy consumption were measured for each aspect of the system. A bash script was created to gather memory utilisation measurements from the free command. Measurements were ensured to be taken from hardware performance counters for both the memory and CPU. Measurements were taken every second during the duration of the program execution.

Unlike in papers such as [36], no budget was available to use a tailor-made system for logging energy consumption measurements in real-time, however it was formulated that quite an accurate testbed could be created using the Adafruit INA219 current sensor breakout board [37]. Energy consumption can be measured using a shunt, or current-sense resistor. A shunt is connected in series so that it carries the current intended for measurement, while a device to measure voltage can be connected in parallel. The Power can then be calculated from Ohm's law as

$$Power = Voltage \times Current \quad (1)$$

Energy is then easily calculated as

$$Energy = Power \times time \quad (2)$$

B. Classification Evaluation

In terms of data preparation, 10-fold cross validation was applied to the dataset before training and testing each of the five classification algorithms, to reduce overfitting. Each algorithm was also tested a number of times to ensure anomalous results were not used.

As expected, from the five classification algorithms both XGBoost and Random Forest performed best. However, the large difference in model size and very small difference in accuracy, show the benefits of memory and size optimisation of XGBoost.

Initially the size of Random forest was thought to be due to the number of trees used. However after closer inspection, it became clear that using a default maximum depth value in the implementation was the main cause for its incredibly large model size. With a maximum depth for each tree in the ensemble set to 3, the model size did dramatically reduce to 703Kbytes, however the accuracy of the model was also reduced to 92%, showing the clear trade-offs in its performance.

Table I: Classification Training Results

Model	Accuracy (%)	Model Size (KBytes)	Training Time (s)
ProtoNN	83.7623	18.720	489.51
Bonsai	93.1414	18.921	345.07
SFSVC	78.2082	344.432	23.48
XGBoost	99.4826	518.424	384.12
Random Forest	99.9071	33106.391	62.87

As can be seen in Table I, Bonsai performed best in the compromise between accuracy and model size. The algorithms ability to split on non-linear decision boundaries may be part of the reason for this. The performance Bonsai also conforms to the theory of tree-based classifiers performing best on the dataset, alongside XGBoost and Random Forest. This may be due to the use of mutual information as a metric for feature selection, allowing the trees clear features to split on.

The most surprising of the five was the under-performance of SFSVC. Despite its very fast training time, SFSVC performs worst in terms of accuracy and created a model size, multitudes larger than both ProtoNN and Bonsai. The lack of accuracy on this dataset may be due to one of SFSVC's advertised advantages: the lack of tuneable parameters. With only a radius size to change, little more could be done to optimise the algorithm on the dataset.

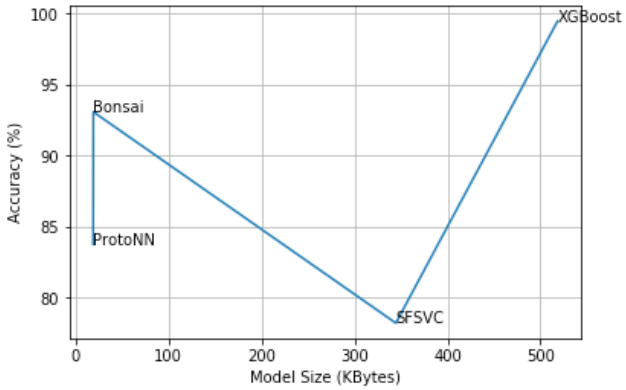

Figure 5: Accuracy vs Model Size

Random Forest was left out of figure 5 above, allowing the accuracy vs model size relationship for the remaining four algorithms to be more apparent. Bonsai and XGBoost offer the best compromise between accuracy and model size, depending on how lightweight the NIDS must be.

The performance of Random Forest, XGBoost and SFSVC were measured on predicting the class of network data taken from the dataset without labels (this was converted from the pcap file format using the feature extraction process created in this project). A script was made to minimise other resource usage during these measurements, particularly ensuring the HDMI port was disabled. Saved models created during the training process were also used to ensure the new testing process was not affected by training time. Unfortunately due to complications, Bonsai and ProtoNN could not be tested under these conditions. Each of the three models were tested a number of times, and the average of these were used.

Table II: Classification Prediction Results

Model	Average CPU Utilisation (%)	Average Memory Utilisation (%)	Latency (s)	Energy Consumption (mJ)
SFSVC	48.70	26.47	4.6	14533.96
XGBoost	49.87	27.01	6.6	18545.64
Random Forest	49.20	26.99	4.8	16907.19

As can be seen from Table II, each of the three algorithms, regardless of size performs quite similarly. This may be due to the relatively simple task of predicting classes from a saved model. Unlike in the training results, XGBoost arguably performs worst here, with the most CPU usage, longest latency and by far, biggest energy consumption. This may be due to optimisation of XGBoost to more traditional computing environments, with more CPUs. Despite the SFSVCs accuracy and model size downfalls, it's smaller and simpler implementation is highlighted by its outright more lightweight performance on the Pi. With its implementation purely limited to the use of the NumPy python library, the implementation is comparatively lightweight.

Although not clearly evident in from Table II, the CPU usage of the three algorithms fluctuated significantly over the course of the measurement period. Figure 6 illustrates this. Of the five CPU measurements taken, the median CPU-Time graph was used for each of the three algorithms.

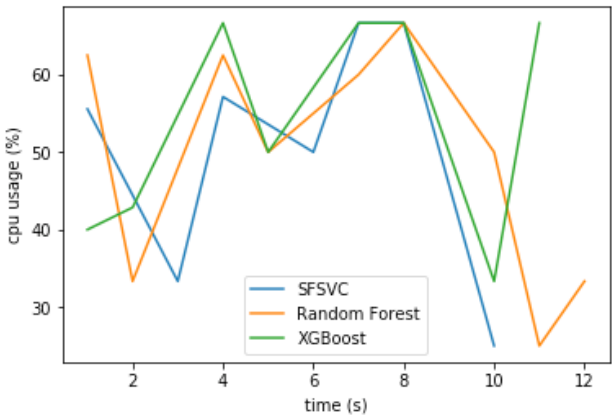

Figure 6: CPU Utilisation of Classification Models

As seen in Figure 6, each of the three models follow a similar pattern. A latency period of three seconds was used, both before and after executing the models. Confirming the findings from Table II, SFSVC uses the least amount of CPU while XGBoost utilises the most. Again, this may be attributed to the simpler implementation of SFSVC.

As previously mentioned, the packet capture and feature extraction elements of the design were also tested under the same metrics. Due to the already quite lightweight performance of the Argus tool for feature extraction, the packet capture elements were tested here. Tshark and tcpdump, two of the popular network packet capture tools were tested to determine which is more lightweight and suited to this application.

Table III: Packet Capture Results

Packet Capture Tool	Average CPU Utilisation (%)	Average Memory Utilisation (%)	Latency (s)	Energy Consumption (mJ)
tshark	38.37	31.78	6.2	39072.00
tcpdump	40.19	29.70	3.0	36908

Overall, tcpdump performs better in this evaluation, with a latency half the time of tshark and a lower level of energy consumption. Conforming to the theory that tcpdump is more lightweight that Wireshark. However, although not shown in

these results, from testing both of the tools, tcpdump had a much larger packet loss rate. A Buffer size of 4MB had to be used for tcpdump. However even with this buffer size twice the size of that of tshark, the packet loss was still quite large. This shows the trade-offs in this situation. With it not being as possible to scale to larger traffic rates or networks with more edge devices, the packet loss rate was not formally evaluated in this context.

One other benefit of tshark, is its ability to identify more protocols, and therefore more anomalous protocols on the network. Ultimately both tools introduce a significant latency into the system.

As previously stated, another version of the overall system was created with bash scripts replacing some of the more cumbersome python implementations of the system. The resource usage of this overall system was also measured, with XGBoost used as the classifier and tshark used for packet capture (with a thirty second capture window). The performance of this is shown in Figure 7 below.

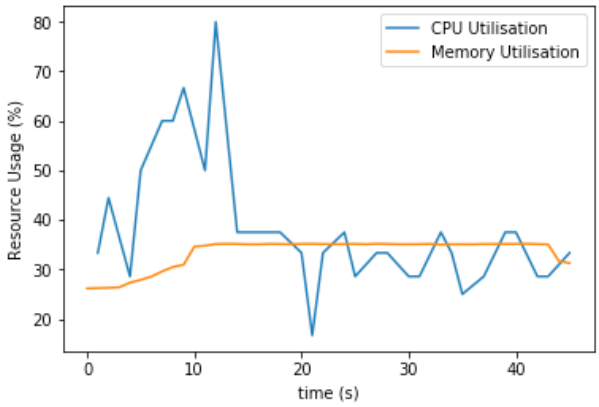

Figure 7: Overall System Resource Usage

A three second window was used in the measurement script, before and after the program was run to give a better visual indication of the performance of the system. The spikes in the first fifteen seconds of runtime can be attributed to tshark beginning the capture process. This clearly levels off as the capture process continues. Around thirty three seconds, the there is a decline in use as the feature extraction process begins. This increases again as XGBoost is used for classification, roughly at thirty eight seconds. Evidently, memory use stays consistent throughout the performance. With only a 2MB buffer used by tshark for the capture process and the classification model being loaded, no major memory intensive tasks are carried out.

The biggest concern overall, as with each part of this system is the latency involved. Despite the thirty second capture window and three second sleep commands used, the system has an overall latency of roughly six seconds. This would need to be improved for more real-time settings. However, a large proportion of this latency can be attributed to tshark, as seen in the individual capture results in Table III.

V. Conclusions & Future Work

In this work, five machine learning classification algorithms were evaluated for the application of network intrusion detection for IoT systems to detect the different classes of botnet behaviour. A number of network transactional flow statistic features were evaluated based on the a number of performance overhead metrics and mutual information, with eight being chosen. Out of the five classifiers, the Bonsai tree algorithm showed the best compromise between accuracy and model size, while the Super-Fast Support Vector Classifier showed the lowest performance overhead for prediction out of the three algorithms tested. Clear trade-offs were also shown between the use of the two packet capture tools.

Future work could expand the testing of this NIDS system to a more realistic IoT setting to test the scalability and performance overhead in terms of a more realistic environment. The performance overhead measurements could also be expanded to ProtoNN and Bonsai. Furthermore, using Morphnet to reduce deep learning based models could be evaluated.

VI. References

[1] D. Bandyopadhyay and J. Sen, "Internet of Things: Applications and Challenges in Technology and Standardization," *Wireless Personal Communications,* vol. 58, no. 1, pp. 49-69, 2011.

[2] K. Angrishi, "Turning Internet of Things(IoT) into Internet ofVulnerabilities (IoV) : IoT Botnets," 13th February 2017. [Online]. Available: https://arxiv.org/abs/1702.03681. [Accessed 29th June 2019].

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

[4] M. Kuzin, Y. Shmelev and V. Kuskov, "New trends in the world of IoT threats," Kaspersky, 18th September 2018. [Online]. Available: https://securelist.com/new-trends-in-the-world-of-iot-threats/87991/. [Accessed 20th June 2019].

[5] N. Chaabouni, M. Mosbah, A. Zemmari, C. Sauvignac and P. Faruki, "Network Intrusion Detection for IoT Security based on Learning Techniques," *IEEE Communications Surveys & Tutorials,* 2019.

[6] Q. Zhu, R. Wang, Q. Chen, Y. Liu and W. Qin, "IOT Gateway: BridgingWireless Sensor Networks into Internet of Things," in *International Conference on Embedded and Ubiquitous Computing*, Hong Kong, China, 2010.

[7] R. Doshi, N. Apthorpe and N. Feamster, "Machine Learning DDoS Detection for Consumer Internet of Things Devices," in *IEEE Security and Privacy Workshops (SPW)*, San Francisco, CA, USA, 2018.

[8] C. Sample and K. Schaffer, "An Overview of Anomaly Detection," *IT Professional* , vol. 15, no. 1, pp. 8-11, 2013.

[9] Raspberrypi, "Raspberrypi.org," Raspberrypi.org, [Online]. Available: https://www.raspberrypi.org/. [Accessed 2 August 2019].

[10] H.-J. Liao, C.-H. R. Lin, Y.-C. Lin and K.-Y. Tung, "Intrusion Detection System: A Comprehensive Review," *Journal of Network and Computer Applications,* vol. 36, no. 1, pp. 16-24, 2013.

[11] C.-F. Tsai, Y.-F. Hsu, C.-Y. Lin and W.-Y. Lin, "Intrusion detection by machine learning: A review," *Expert Systems with Applications,* vol. 36, no. 10, pp. 11994-12000, 2009.

[12] T. Mehmood and H. B. M. Rais, "Machine Learning Algorithms In Context Of Intrusion Detection," in *International Conference On Computer And Information Sciences*, Kuala Lumpur, Malaysia, 2016.

[13] S. Hao, N. A. Syed, N. Feamster, A. G. Gray and S. Krasser, "Detecting Spammers with SNARE:Spatio-temporal Network-level Automatic Reputation Engine," *USENIX security sympo- sium,* vol. 9, 2000.

[14] A. Sivanathan, D. Sherratt, H. H. Gharakheili, V. Sivaraman and A. Vishwanath, "Low-cost flow-based security solutions for smart-home IoT devices," in *IEEE International Conference on Advanced Networks and Telecommunications Systems*, Bangalore, India , 2016.

[15] G. Kambourakis, C. Kolias and A. Stavrou, "The Mirai botnet and the IoT Zombie Armies," in *IEEE Military Communications Conference*, Baltimore, MD, USA , 2017.

[16] J. Margolis, T. T. Oh, S. Jadhav, Y. H. Kim and J. N. Kim, "An In-Depth Analysis of the Mirai Botnet," in *International Conference on Software Security and Assurance*, 2017, Altoona, PA, USA .

[17] Y. e. a. Meidan, "N-BaIoT—Network-Based Detection of IoT Botnet Attacks Using Deep Autoencoders," *IEEE Pervasive Computing ,* vol. 17, no. 3, pp. 12-22, 2018.

[18] C. D. McDermott, F. Majdani and A. V. Petrovski, "Botnet Detection in the Internet of Things using Deep Learning Approaches," in *International Joint Conference on Neural Networks*, Rio de Janeiro, Brazil , 2018.

[19] J. e. a. Su, "Lightweight Classification of IoT Malware based on Image Recognition," in *Computer Software and Applications Conference*, Tokyo, Japan , 2018.

[20] 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, pp. 779-796, 2019.

[21] A. A. Gendreau and M. Moorman, "Survey of Intrusion Detection Systems towards an End to End Secure Internet of Things," in *Future Internet of Things and Cloud*, Vienna, Austria , 2016.

[22] J. Arshad, M. A. Azad, K. Salah, W. Jie, R. Iqbal and M. Alazab, "A Review of Performance, Energy and Privacy of Intrusion Detection Systems for IoT," *Journal of Cloud Computing: Advances, Systems and Applications,* vol. 7, no. 21, 2018.

[23] T. Szydlo, J. Sendorek and R. Brzoza-Woch, "Enabling machine learning on resourceconstrained devices by source code generation ofthe learned models," in *International Conference on Computational Science*, Wuxi, China, 2018.

[24] A. e. a. Gordon, "MorphNet: Fast & Simple Resource-Constrained Structure Learning of Deep Networks," in *Computer Vision and Pattern Recognition*, Salt Lake City, UT, USA, 2018.

[25] C. Gupta and e. al., "ProtoNN: Compressed and Accurate kNN for Resource-scarce Devices," in *International Conference on Machine Learning*, Sydney, Australia, 2017.

[26] D. Dennis, "Tensorflow ProtoNN Examples," Guthub, 31 Mat 2019. [Online]. Available: https://github.com/microsoft/EdgeML/tree/master/tf/examples/ProtoNN. [Accessed 15 June 2019].

[27] A. Kumar, S. Goyal and M. Varma, "Resource-efficient Machine Learning in 2 KB RAM for the Internet of Things," in *International Conference on Machine Learning*, Sydney, Australia, 2017.

[28] A. Kusupati, "EdgeML Bonsai on a sample public dataset," Github, 21 June 2019. [Online]. Available: https://github.com/microsoft/EdgeML/tree/master/tf/examples/Bonsai. [Accessed 25 June 2019].

[29] R. Dogaru and I. Dogaru, "Optimized Super Fast Support Vector Classifiers Using Python and Acceleration of RBF Computations," in *International Conference on Communications*, Bucharest, Romania, 2018.

[30] R. Dogaru and I. Dogaru, "A super fast vector support classifier using novelty detection and no synaptic tuning," in *International Conference on Communications*, Bucharest, Romania, 2016.

[31] R. Dogaru, "Super_Fast_Vector_Classifier," Github, 18 May 2019. [Online]. Available: https://github.com/radu-dogaru/Super_Fast_Vector_Classifier. [Accessed 22 June 2019].

[32] T. Chen, "XGBoost Documentation," XgBoost, [Online]. Available: https://xgboost.readthedocs.io/en/latest/. [Accessed 15th July 2019].

[33] Scikit Learn, "sklearn.ensemble.RandomForestClassifier," Scikit Learn, [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html. [Accessed 30 July 2019].

[34] N. e. a. Koroniotis, "The BoT-IoT Dataset," The center of UNSW Canberra Cyber, 14 November 2018. [Online]. Available: https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA-NB15-Datasets/bot_iot.php. [Accessed 11 June 2019].

[35] Argus, "The Argus Archive," QoSient, 14 January 2019. [Online]. Available: https://qosient.com/argus/. [Accessed 15 July 2019].

[36] F. Kaup, P. Gottschling and D. Hausheer, "PowerPi: Measuring and Modeling the Power Consumption of the Raspberry Pi," in *IEEE Conference on Local Computer Networks*, Edmonton, Canada, 2014.

[37] Adafruit, " INA219 High Side DC Current Sensor Breakout - 26V ±3.2A Max," Adafruit, [Online]. Available: https://www.adafruit.com/product/904#technical-details-anchor. [Accessed 18 July 2019].