Generating Synthetic Data for Real World Detection of DoS attacks in the IoT

Luca Arnaboldi and Charles Morisset

School of Computing, Newcastle University, Newcastle upon Tyne, UK
{l.arnaboldi,charles.morisset}@ncl.ac.uk

Abstract. Denial of service attacks are especially pertinent to the internet of things as devices have less computing power, memory and security mechanisms to defend against them. The task of mitigating these attacks must therefore be redirected from the device onto a network monitor. Network intrusion detection systems can be used as an effective and efficient technique in internet of things systems to offload computation from the devices and detect denial of service attacks before they can cause harm. However the solution of implementing a network intrusion detection system for internet of things networks is not without challenges due to the variability of these systems and specifically the difficulty in collecting data. We propose a model-hybrid approach to model the scale of the internet of things system and effectively train network intrusion detection systems through bespoke datasets generated by the model, able to predict a wide spectrum of attacks. The proposed model is able to mimic an attackers behaviour, and as demonstrated by an experiment construct more predictive datasets at a fraction of the time of other more standard techniques.

1 Introduction

A Denial of Service (DoS) attack targets the availability of a device or network [19], with the intent of disrupting system usability. The most common method is referred to as Flooding DoS [22], and may be used as an attempt to deplete the devices' resources including memory, bandwidth and/or battery. A DoS attack against an Internet of Things (IoT) network has the potential to be significantly more detrimental than one against a standard network. This increased vulnerability is due in part to the low computational power and battery power characteristic of IoT devices [28].

The extant literature has delineated several potential approaches that may be effective in the mitigation of a DoS attack [29]. They widely speaking fall into two categories, host based (e.g. Client Puzzles) which puts the computational effort on the device and network based (e.g. firewall) which offloads the computational effort to a remote server or more powerful device within the system. However many of these approaches may not scale well in the IoT as computational power, heterogeneity and the large scale of these systems are all limiting factors that deplete the available choices. One approach that sidesteps many

of these standard detriments is a Network Intrusion Detection System (NIDS) bespoke to the IoT system to protect. NIDS are monitors placed on the network that analyse incoming traffic to detect attacks and/or unwanted traffic. They are trained using system behaviour data and use these patterns to make the detection.

Organizations and researchers alike have widely recognised the advantages of adapting NIDS as the norm to monitor against DoS attacks on their systems [23]. Standard approaches used to train NIDS include using a database of known attacks (*misuse detection*) and testing systems to create a "benchmark" behaviour and flag any anomaly as a potential attack (*anomaly detection*) [21].

Implementing a NIDS within an IoT network however faces multiple challenges. Firstly, it is usually challenging to establish a benchmark behaviour in dynamic IoT systems as devices may constantly shift, new devices might join and behaviours might change [13], which might prevent using anomaly detection. Secondly, protocols can vary from one network to another, which necessitates data collection to be bespoke to an individual system [14]. Thirdly, a misuse detection can be time consuming to enforce, since collecting data unique to a system and for each attack is time consuming [21] and some system changes can require data (or part of the data) to be collected from scratch (e.g. interactive smart homes where devices can change frequently [1]).

To address the second and third challenges, we present a novel modelling approach. In brief, our model consists of a collection of Markov Decision Processes (MDP), representing the IoT network, the attackers, and some processes monitoring the security metrics under consideration. A trace of the model (corresponding to a sequence of actions of the different MDPs) should match a trace of the actual system, and vice versa, such that it becomes possible to train a NIDS for the actual system on the traces of the model. The main strengths of our approach is the ability to easily represent various configurations for the IoT network as well as multiple types of attackers. MDPs have some key advantages: they have substantial tool support such as PRISM Model Checker [17] and that they rely on probabilities and non-determinism to recreate systems behaviour. Through non-determinism we can create traces of behaviour that mimic attacks on systems by assigning rewards to successful behaviour. Our results highlighted that through this methodology we were able to consistently produce synthetic datasets that resulted in more accurate NIDS (detecting attacks on real world systems) and that could be trained in a fraction of the time.

The core contributions of this paper are 1) A model of an IoT system that enables the generation of synthetic data sets of network behaviour 2) Modelling of attack behaviour against a system to train a real world NIDS 3) A quantitative analysis and validation of this model against a real world implementation of the same system to validate our methodology.

The paper is split into the following sections; In Section 2 we discuss the problem overview in more detail, provide an overview on our approach and build on existing literature; In Section 3 we introduce our IoT system model and attacks model that generates the network behaviour; In Section 4 we highlight

our assessment and evaluation methodology; Section 5 provides an analysis of our results and section 6 concludes and discusses future work.

2 Formulation of Problem

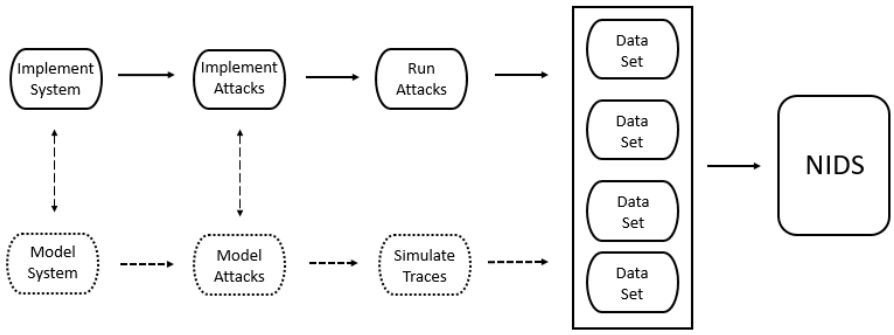
In order to successfully train a NIDS for a bespoke system a security professional needs to collect large quantities of data. This data is then used to train a NIDS, which is in essence a Machine Learning classifier that can establish whether a set of network packets entering the system is malicious. The problem with this is that to gather this data one has several options each with several drawbacks, option 1) you make use of known attack data sets however these might not exist for their network configuration (e.g. if using differing connectivity and communication protocols) option 2) make use of existing NIDS (e.g. Signature-based Intrusion Detection for Networks (SNORT) [25]) however this produces large amounts of false positives and doesn't cater to unknown attacks, or option 3) make use of an exploit database and simulate attacks on your own system. This approach is by far the most precise [21] as it allows to search for all attacks known to your system and construct a data set which is bespoke and unique to your system. Whilst this approach produces the best suiting dataset it has some major drawbacks. Firstly, one must find and implement the attacks, which is a difficult process that might take a very long time. The second drawback is that one would need to cause major disruptions to one's own network by simulating the attacks which might hamper work and is therefore inefficient. What our modelling approach does is the ability to have the same level of customization and precision cutting out the man hours of implementation and disruption to the real word system but adding several other desirable features such as behaviour predictions and the ability to generate large quantities of behavior data with ease.

A standard way to accumulate data is through use of log files, through these logs a security professional or NIDS can spot malicious behaviour such as DoS attacks or malicious injections. In order to cross validate the model output for consistency and to be able to analyse system behaviour, the output of the model needs to fit the template of these files as closely as possible. This was, in part, a way to easily check for matching behaviour between the model and the real system. One of the many difficulties in detecting attacks on systems through the use of NIDS, is that one cannot (easily) predict potential attacker behaviour, or rather it is very difficult to classify an attack if it doesn't seem like past attacks and there is no previous data that looks like it. Below is a overview of the full problem and a small case study to validate our approach:

Scenario overview - Our Scenario features four characters each representing different parts of the proposed experiment. Device A and Device B each represent two IoT devices in a system, specifically low powered IoT sensors. Device A and Device B are intercommunicating and they transmit data back and forth.

Through this we create a benchmark of system behaviour. There is a third component, Device E (representing a potential attacker), Device E is a malicious device of higher computational power than the other two parties, the fact device E is malicious remains unknown to system administrators as its identity is obfuscated. Device E has the capabilities of implementing a DoS attack able to take down the other two devices. The system administrator observes this behaviour and establishes a NIDS to monitor these attacks and prevent their occurrence in the future. The NIDS must be able to predict when these attacks are about to occur accurately to avoid false positives and false negatives. The problem is two fold; disallow Device E from taking down the system (stop DoS attacks) but also not stopping legitimate traffic based on a false positive of a potential attack. The system administrator's first approach uses data from previous attacks to train the NIDS. However, training on these datasets leads to issues detecting patterns of behaviour that differ from past behaviour (unknown attacks), causing inaccurate predictions and false positives/negatives. The second approach they use is: taking the knowledge of the systems behaviour, they simulate further data by creating a model that mimics the systems behaviour. This model matches the behaviour of the system and can be run in parallel to the real system without disrupting uptime or changing the real system. The outputted data can then be used to create a more accurate solution to the problem that is tailored to our specific scenario, able to detect DoS attacks without running attacks on the system (for more real data) nor waiting for more attacks to take place and using that data.

Fig. 1. Proposed approach allows for the ability to generate synthetic datasets to be used to train a real world NIDS to detect future attacks.



2.1 DoS Attacks on IoT Systems

DoS attacks have long been one of the most common and dangerous threats in any internet system. these attacks become even more dangerous as the IoT spreads across a vast amount of spectrums and parts of life including safety

critical and potentially life endangering ones such as IoT healthcare [15] and ITS [24].

The extant literature highlights several new DoS attacks against IoT system taking advantage of unique qualities and IoT infrastructures [18, 26, 10]. One such attack, battery drain attack focuses on exhausting the devices battery power as replacing it might be costly, difficult and lead to extensive periods of downtime. These kinds of attack are very subtle as the behaviour of the attacker might not necessarily mimic more common attacks such as pure flooding, they attempt to find battery intensive operations (not necessarily malicious) and repeat them until the device is out of power.

This is only one specific example of the literature cited above. What they do have in common is that they are specialised with the intent of disrupting IoT devices and as such many of the current detection systems do not account for them as they are very unique to specific setups [26]. The literature highlights that there is a constant evolution of attacks, as can be seen using resources such as ExploitDB [3]. When filtering for IoT attacks we can observe that there is a huge increase in the spectrum of attacks.

These upwards trends in combination with the expansion of the IoT across various field makes a good argument for a simple way to observe the impact of these attacks. Our formalisation would allow for intuitive means to observe and quantify these attacks as well as better defend these systems by generating network behaviour bespoke to them.

2.2 Network Intrusion Detection Systems

The growing use of Internet service in the past few years have facilitated an increase in DoS attacks. Despite the best preventative measures, DoS attacks have been successfully carried out against various companies and organizations enforcing the need for better prevention/detection mechanisms. This is partially due to the vast new avenues of attack (often unique to IoT) that signature based schemes such as SNORT [25] simply struggle to detect.

Further work attempts a more scalable approach that models behaviour of a network (stationary or non stationary) and labels abnormal packets as a potential anomaly [20, 9]. Limitations of this approach are a large number of false positives as well as lack of information regarding the attack (e.g. the specific vulnerability the attack relies on) as opposed to a signature based NIDS that is able to tell you exactly what rule is broken.

The work we present allows for a mixture of these approaches tackling the limitations of both work. By modelling behaviour of a system, we can detect any anomaly similar to the second approach however by modelling various attacks we can also provide accurate data of the system behaviour whilst being attacked, allowing for less false positives. To be able to predict "unknown" attacks our modelling approach uses a stochastic attacker that attempts different behaviour allowed by the system policy. Using this data we can create signatures for potential attacks and simulate an attacker probing the system.

2.3 Modelling IoT Systems

Several papers address modelling IoT, adopting various different approaches. Authors of [7, 30] have worked on modelling a specific IoT protocol on the transport layer, looking at MQTT and CoAP respectively. Fruth [12] examines various properties of a Wireless Network protocol including connectivity and energy power through PRISM, evaluates it on a Wireless Sensor Network and evaluates the battery drainage of certain randomized protocols. The authors of [6] take this work a step forward and model basic DoS attacks through PRISM and look at the effectiveness of different attack strains and mitigation techniques.

We combine these approaches to recreate an accurate representation of system behaviour and represent a wide range of DoS attacks, PRISM has been widely used as a excellent method to evaluate and verify models of IoT systems and protocols, combining these two models by adapting both the system models and the protocol models we successfully model the behaviour and general properties of a bespoke IoT system. We then use the inbuilt verification capabilities to ensure correctness relative to mimicking system behaviour by establishing benchmarks and tests and then use the simulation capabilities of prism to predict behaviour and simulate attacks against the verified model.

3 IoT System Model

The model we present makes use of MDP's and their ability to produce large amount of system traces, to model the behaviour of the IoT system and bespoke attackers. Using our model, one can simulate different power of attacks (e.g. a botnet with multiple devices), as a factor of messages sent and predict the impact on the system (e.g. decrease in throughput, time to system failure etc). Each of the MDP components of the model is based on measurements from the actual system (e.g. the time required by a specific device to process a message). The model represents key characteristics of an IoT device to infer specifics of a real world system (e.g. time to drain of battery under a specific attack) and also generates traces of behaviour of the system under attack, being able to cover potential as well as known attacker behavior. The specifics in question for the devices are memory and battery power, since each device will have unique battery levels and memory that could easily be overwhelmed by an attacker. The attacker MDPs are manually defined to match the behaviour of real world attacks. A particular strength of our approach is the ability to find the optimal path of attack through an *MDP adversary* [16] (although we still need for all possible attacks to be modelled first, it can then find the optimum strategy to take down a bespoke system).

3.1 IoT Device Model

To model low power IoT devices we make the following assumptions: at the communication layer low power IoT devices capabilities are generally limited

to reading/receiving and passing on/distributing data [13] independent of the protocol used, that the bandwidth is independent of the device, and that the battery drainage is linear in respect to current [2]. The action labels of the MDP are used in to represent the behaviour of a device and to synchronize with other parts of the system. The device has an associated monitor that synchronizes on each action and guards for the current values of battery and memory.

In the model a monitor is non blocking, the calculations for battery are based on increase in Amperes at time T based on processing of messages. The larger the inflow of messages (due to attacker processing power) the larger the drain on the battery [2, 6, 5]. The monitor allows to calculate traces of executions that lead to battery drainage and/or memory exhaustion of the devices.

A Markov decision process (MDP) is a tuple $M = (L, \mu, \Gamma, \delta, r)$ where L is a finite set of states, $\mu \in D(L)$ is an initial probability distribution over states, Γ is a finite set of actions, $\delta : L \times \Gamma \to D(L)$ is a probabilistic transition function that assigns to each pair of states and actions, a probability distribution over successor states and r is a reward functions that assigns a reward value to each action γ [11].

We describe the behaviour of an MDP as a device sending/receiving messages infinitely many times. In the first state, the device starts in state ℓ with probability $\mu(\ell)$. Each time the device is activated if the device is in the state ℓ and the device chooses the action $\gamma \in \Gamma$, then the device moves to the successor state ℓ_i chosen with probability $\delta(\ell, \gamma)(\ell')$.

To implement the case study we made use of PRISM Model Checker [17] to abstract systems of IoT devices. The PRISM tool also allows use of statistical model checking, a technique which is particularly effective since modelling multiple devices exacerbate the state space immensely. We built the device based on two core requirements accurate representation of processing power (of both an attacker and the systems device) by means of messages per second and accurate representation of battery and memory and respective drainage under DoS strain.

Running Example - Device Setup : We set up a small IoT network in the lab to test out the effectiveness of our model in creating synthetic datasets. For the sake of testing we kept the setup simple to display the tool as the thing that needs to scale and not the system. Once the simple model is created it is trivial to add more (similar) devices, whilst implementing a new system in the real world can be very time consuming. We implemented a sensor network consisting of two devices. Each device had the following *actions*; they took sensor readings and then could *send* it to the other device at any time; they could also *request* the sensor data from the other device at any point. The devices used simple HTTP protocol for communication, and the behaviour was stored in Apache log format.

To accurately represent the devices and to create *smart* attackers several measures needed to be obtained. Both devices were equipped with a Mh3500 battery, by default however sensor devices have no means to calculate remaining battery power, so we utilized a tool to predict battery drainage [2]. Results

from the tool tested correctly on the real system and we therefore took its accuracy as an assumption. This was used in combination with a variance we introduced to represent attack intensity and change to current. To model this we made a basic assumption that the devices are on constantly. We argue this is a correct assumption as due to our attack the device is constantly in log mode and therefore never in sleep mode. Beyond this assumption we calculated time it took to send a message/log a message, baseline battery usage, percentage increase in battery usage under different DoS strains (taken this value and dividing it by messages processed for second) and battery drain per message and calculated in $calc_s$ and $calc_r$ as per **Table 1**. The behavior is broken down as follows:

Components:
Devices = $\{Device_A, Device_B\}$
Monitors = $\{Monitor_A, Monitor_B\}$
Attacker = {Attacker}
Behaviour:
$Device_i = idle_i \rightarrow STOP \ \sqcap \ send_i \rightarrow STOP \ \sqcap \ rec_i \rightarrow STOP$
$Monitor_i = send_i \rightarrow calc_{s_i} \rightarrow STOP \ \sqcap \ rec_i \rightarrow calc_{r_i} \rightarrow STOP$
Attacker = $\ \forall \ i \in Devices \Rightarrow selAttack \rightarrow selDevice \ \sqcap \ send_i \rightarrow STOP$
System:
Attacker $|[\{sending_i\}]|$ $Device_i$ $|[\{sending_i, receiving_i\}]|$ $Monitor_i$

Table 1. Details of the monitor calculations and model setup, Γ is the set of actions, t is the time, M is the memory, M_c is the current memory load, B is the battery, B_c is current battery level, $S_{A/B}$ is sending message from A or B, $R_{A/B}$ is the receiving message from A or B, n is a message, Q_i is a queue of messages in A or B, Δ A is the variance in current

	Device A Send	Device A Receive	Device B Send	Device B Receive
γ	$[send_A]$	$[rec_A]$	$[send_B]$	$[rec_B]$
t	$t_0 + t_{SA}$	$t_0 + t_{RA}$	$t_0 + t_{SB}$	$t_0 + t_{RB}$
M	$M_c - len(Q_i)$	$M_c + len(Q_i)$	$M_c - len(Q_i)$	$M_c + len(Q_i)$
B	$B - (S_A(n) \rightarrow \Delta A)$	$B - (R_A(n) \rightarrow \Delta A)$	$B - (S_B(n) \rightarrow \Delta A)$	$B - (R_B(n) \rightarrow \Delta A)$

These core measurements were used to quantify the effectiveness of the attacker. We theorize that through this synthetic data, the NIDS will train a more accurate predictor especially against unknown attackers. Doing a similar approach without the model would require attacking your own system and implementing an attack to collect data.

3.2 Attacker Model Implementation

An attack synchronizes with a subset of actions of the device, when an attacker synchronizes on the device, the monitor will synchronize on that action and

calculate the respective drainage. The monitor keeps track of all these measurements for its respective device. Implementing the model in a tool like PRISM allows us to make use of Probabilistic Control Tree Logic (PCTL) [8] to calculate various conditions of pertinence to the system, to compute the optimal attack path, and to simulate traces of actions.

Given a *policy* regulating the behaviour (corresponding to an attack type) we allow for any action to take place at any point, this can be combined with a set of rules to find the trace of behaviour that allows to follow all the rules and yet still drain the battery as quickly as possible within these restrictions. By applying reward to each message associated with time (translating to time per message) we can find the optimal method to drain the battery in the least amount of time possible.

Likewise we can find generate traces of less detectable attack by associating an anomaly score to an action and therefore keeping the anomaly behaviour to a minimum whilst still optimizing time, leading to various different scenarios that can be run. In essence, we assume that if the attack is repeated in the same manner (e.g. from the same url) multiple times in a row it will make it easier to detect, given this knowledge we can implement a counter, if this counter reaches a certain value a negative reward is applied. Each different combination of variables has a different counter so by changing message structure you reduce your chance of getting a negative reward. In the case where the value is five, and there are altogether twelve different combinations of messages to send, structured as per explained in the running example (HULK) and it takes one hundred messages to hypothetically drain the battery of a device, if the attacker used the same message continuously the negative reward would be applied a total of twenty times, however if it continuously alternated it would be able to take down the system only triggering the negative reward once. This leads to non predictable behavior and the ability to maximise the effectiveness of the attacker.

The non-determinism in our model is achieved through the *Attacker* strategy, we implement a reward structure *time* and find the optimum path. In PCTL it is written as $R\{\text{“}time\text{”}\}min =?[\ \mathrm{F}\ power = 0\ ]$ or the minimum time for the variable power (referring to battery levels) to reach 0. The value “time” is a variable calculated by the time for a single message to be sent by the attacker and cumulated for each message sent before the power reaches 0 calculated in microseconds and the power drain is calculated by the formula mentioned in section 3.1. These Reward structures allow for simulated attack strategies that an hypothetical attacker might make to take down the modelled IoT system.

Running Example - To validate the model we implemented a common DoS attack both in the real world and in the model. Our attack of choice was HULK, a DoS attacking tool which relies on several obfuscation techniques in order to not be spotted whilst still outputting intense strain enough to take down systems very quickly [4]. The attack specifies it has the following properties: 1) obfuscation of Source Client - this is achieved by using a list of known User Agents, and for every request that is constructed, the User Agent is a random

value out of the known list, 2) reference forgery - the referer that points at the request is obfuscated and points into either the host itself or some major pre-listed websites, 3) stickiness - using some standard Http command to try and ask the server to maintain open connections by using Keep-Alive with variable time window and 4) unique transformation of URL - to eliminate caching and other optimization tools, they crafted custom parameter names and values and they are randomized and attached to each request.

The tool was able to take down a web server within minutes from just a single host. Seeing as IoT devices will have less capabilities than any web server we hypothesized that this would be a good attack to use as its properties make for a good dataset that is not straightforward to detect. These properties and obfuscations led to different combinations of message structure that we used in the non-deterministic attacker.

The reward structure or monitor can allocate a time value to each message sent by the attacker (the lower the value the larger the processing power). So by taking the time it takes per each message we accurately measure how many messages can be sent within a time period as well as evaluating its accuracy in respect to the real world based on our test attacker. We also assumed a lossless channel and therefore every message sent by the attacker will reach the device.

4 Verification & Evaluation

To evaluate the effectiveness of the models we evaluated and compared the running example, with a real world IoT system. The verification was on the following basis, the model's behaviour needed to match the real world system and we validated the output data with a naive approach as well as the standard approach described in Section 1.

To verify our model and its output dataset we ran experiments comparing it to two further datasets. The first approach was the state of the art approach, we implemented the system of devices and the real-world attack and collected data across a period of 12 hours (RWD). This constituted the first dataset and the approach to beat.

The second approach was a naive approach, we constructed a synthetic dataset using stochastic parameters and generating an ad-hoc dataset of our devices behaviour (ND). The created dataset had the same components as our tested system and the same protocols however the packets were constructed randomly and the class labelling was ad hoc. The intent of this was to give a comparison of a different synthetic dataset as well as learn more about how the classifiers worked. We can easily show that given the choice to use a synthetic dataset our approach is better than a naive approach. This second dataset represented an alternative approach with some advantages over the state of the art, namely the speed of it and the ease of construction however we theorize it will produce several false negatives and train a weak predictive model.

And finally, we followed our proposed approach (MD): modelling the system to mimic the behaviour and construct *smart* attack behaviour whilst still

mimicking the test attack. We theorized that this would create a stronger predictive model able to classify a wider range of attacks . As our dataset relies on stochastic events and actions we created three datasets from this approach and evaluated each one to benchmark its effectiveness and taking a mean score, furthermore whilst our model is able to recreate very large datasets very quickly we choose to keep the dataset size uniform across the initial experiment to get a fair comparison against the other two datasets.

The second experiment we ran was to test the effectiveness in recreating large datasets. We used deep learning classifiers catered to large datasets and created a much more efficient NIDS purely through synthetic data. One of the core strengths of our approach is that the datasets are very easy to generate and we theorize that this in combination with our *smart* attackers will lead to the ability to train better performing NIDS.

To evaluate our data we implement a Python framework that runs through the various steps: data processing, standardization, training and testing against unknown attacks. This automatic framework evaluates and compares the five datasets. It is implemented in Python using the scikit-learn machine learning libraries.

Data Processing & Standardization - To allow for data to be interpreted by machine learning algorithms it needs to go through a process of standardization. This is often due to categorical non numeric features or continuous features. The data provided by most if not all internet protocols is categorical (e.g. agent names and method calls). As such, in order to evaluate it we first needed to go through an initial phase of pre-processing. The intent of pre-processing is to render the data machine readable whilst preserving patterns. The process we adapted was the process of binarisation. What binarisation achieves is allocating a numeric value to each unique feature for example if dealing with HTTP codes GET would become 0001, POST 0010, DELETE 0100 and PUT 1000. This allows for the features to maintain their patterns and their predictive power and be used normally. This initial step was applied to both the real world dataset and the naive synthetic dataset. This step was however not required for the model dataset as it already produced numeric features rather than categorical ones for efficiency.

Classifiers - The classifiers we implemented represented the NIDS, we choose to use two separate classifiers to get a better evaluation of the results. Each dataset was used to train two NIDS, and then all the NIDS were tested against a new dataset of attack to establish their predictive power and the strength of the datasets.

The first classifier we implemented was Multi Layer Perceptron (MLP) Neural Network. An MLP consists of at least three layers of nodes. Except for the input nodes, each node is a neuron that uses a non-linear activation function [31]. MLP utilizes a supervised learning technique called back propagation for training. Its multiple layers and non-linear activation distinguish MLP from a linear

perceptron. A linear perceptron is a function that can decide whether an input, represented by a vector of numbers, belongs to some specific class or not, combining several together in an MLP and adjusting the functions and weights you build a statistically accurate classifier. The result is a non linear perceptron that is able to classify non linear classes.

The second classifier used was a Decision Tree Classifier. A decision tree is a decision support tool that uses a tree-like graph or model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm that only contains conditional control statements [27]. Decision tree learning uses a decision tree (as a predictive model) to go from observations about an item (represented in the branches) to conclusions about the item's target value (represented in the leaves). Tree models where the target variable can take a discrete set of values are called classification trees; in these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels. Through a built decision tree based on the constructed rules it will be able to predict data and run it until it reaches an end node corresponding to a class (either DoS attack or normal behaviour).

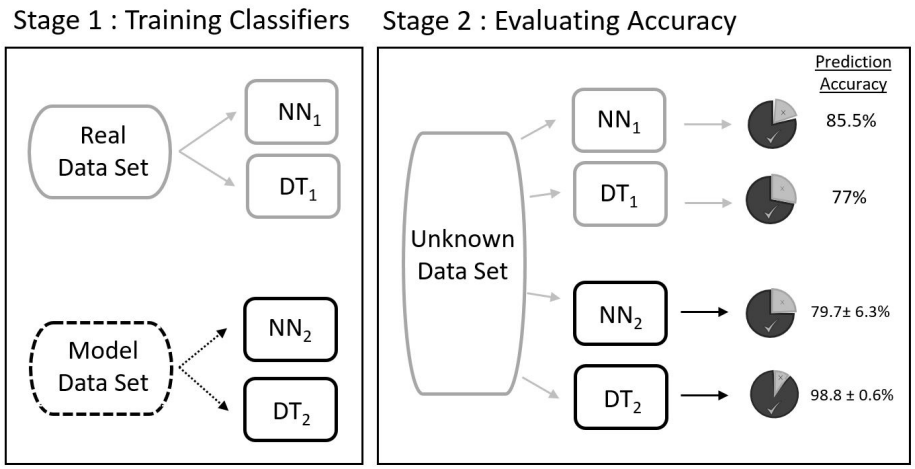
5 Results

Following the evaluation criteria in section 4 and recreating the model described in section 3, we generated and tested three synthetic datasets against our benchmarks of the naive approach and the state of the art. Beyond the accuracy of the results, we also make an argument for feasibility of this approach and re usability. As highlighted in **Fig. 2** The results were acquired by initially training two classifiers for each dataset (real and synthetic), these were trained with 20,000 samples of which 10% were attacks. The classifiers were then evaluated on an unknown and unlabelled real world dataset of 100,000 samples of which 20% were attacks (of two different unknown types). The classifies then attempted to label the new dataset to predict which ones were attacks.

The neural network results on the real world dataset proved to be very accurate with a 85.5% prediction accuracy, on the other hand the the model synthetic data whilst still high suffered from some degree of variance ($79.7 \pm 6.3\%$). What was of most interest however was the predictions outputted by the naive dataset of 0.9% this combined with the relatively inconsistent results of the synthetic dataset ($\pm 6\%$) make a case for over fitting. Over fitting is the scenario in which a model is trained so specifically to the training data that it is no longer classifying DoS attacks and normal behaviour of the system but rather focusing solely on the training data and learning on patters unique to the dataset not the system. This is quite common in Neural Networks as they perform best with very large quantities of data [31] which for this part of the experiment we did not have.

The final classifier relied on constructing trees of rules that would fit to the dataset, contrasting to neural networks decision trees do not suffer from the same inadequacy and do not necessarily need large amounts of data. This was

Fig. 2. Four Classifiers are trained to predict DoS attacks, two with a synthetic dataset and two with real world data, they are then tested for how well they can predict attacks on unknown data.



mirrored by the results, as the model datasets all performed to very similar standards and the added randomness traces which might have disrupted the NN made for a more ample rule set resulting in near perfect predicting power in the model dataset (98.8 ± 0.6). The real world data which did not look at the possibility of random behaviour only achieved 77% accuracy and the random dataset had a predictive power of near 50% as expected.

We observed that our approach of using non-determinism to recreate attack traces was particularly effective for the rule based classifier however led to disruption during the back-propagation process of the NN, as non standardized data can create uneven results in NN if weights of the different perceptrons are confused. However we can also attribute the distortion to a relatively small dataset as such we theorized that this would not be the case if we made use of a very large synthetic data set. Using our model we quickly created a larger dataset of traces (100,000 samples) and retrained a new NIDS using MLP and then used that to predict a new dataset of real world attacks. This time using the much larger dataset the results were a lot more accurate (97.1%) than previously, confirming our hypothesis.

As highlighted by this example our model has one key advantage over the traditional state of the art. Data generation is fast and efficient. If we wanted to improve the training of the NIDS used on the real world data set to a similar level of accuracy it would take several days of data collection and consumption of resources (electricity, system downtime etc). We argue that whilst the initial effort of creating a model might be time consuming and perhaps not as intuitive for a potential system administrator, the phase of dataset generation makes up for this effort both for speed and predicting power of the NIDS.

6 Conclusion & Future work

Our case study and proposed methodology has shown very promising results. We have shown that generating synthetic datasets of DoS attacks in IoT networks through this tool is both effective and efficient. We believe that the ability for this approach to scale easily to multiple devices and protocols in combination with its strong predictive power makes a very good argument for its usage across various IoT networks. Our argument for scalability of this approach is two fold, firstly it scales well in terms of costs as you can make assessment prior to implementing the system and secondly we can bypass several of the downsides of verification (in terms of state space) as we focus on simulation.

In this paper we included a case study of a single attack which worked very well. Our future work envisions the ability to model further attacks from a database to create an extensive set of attacks to create a much more predictive dataset. We envision that the ability to relatively easily plug and play any IoT system in combination with this corpus of attacks we aim to implement, could turn into a tool that generates synthetic dataset of attack to train bespoke NIDS for any IoT system.

References

1. Beolink smarthome, https://www.bang-olufsen.com/en/solutions/beolink-smarthome
2. Calculating battery life in iot applications, http://uk.farnell.com/calculating-battery-life-in-iot-applications#calculator
3. Offensive security's exploit database archive, https://www.exploit-db.com/
4. Hulk, web server dos tool (Feb 2013), http://www.sectorix.com/2012/05/17/hulk-web-server-dos-tool/
5. Abbas, Z., Yoon, W.: A survey on energy conserving mechanisms for the internet of things: Wireless networking aspects. Sensors 15(10), 24818–24847 (2015)
6. Arnaboldi, L., Morisset, C.: Quantitative analysis of dos attacks and client puzzles in iot systems. In: Security and Trust Management - 13th International Workshop, STM 2017, Oslo, Norway, September 14-15, 2017, Proceedings. pp. 224–233 (2017)
7. Aziz, B.: A formal model and analysis of an iot protocol. Ad Hoc Networks 36, 49–57 (2016)
8. Baier, C., Katoen, J.P., Larsen, K.G.: Principles of model checking. MIT press (2008)
9. Bhuyan, M.H., Bhattacharyya, D.K., Kalita, J.K.: Network anomaly detection: methods, systems and tools. IEEE communications surveys & tutorials 16(1), 303–336 (2014)
10. Buennemeyer, T.K., Gora, M., Marchany, R.C., Tront, J.G.: Battery exhaustion attack detection with small handheld mobile computers. In: Portable Information Devices, 2007. PORTABLE07. IEEE International Conference on. pp. 1–5. IEEE (2007)
11. Doyen, L., Massart, T., Shirmohammadi, M.: Synchronizing objectives for markov decision processes. arXiv preprint arXiv:1102.4121 (2011)
12. Fruth, M.: Formal methods for the analysis of wireless network protocols. Oxford University (2011)

13. Gubbi, J., Buyya, R., Marusic, S., Palaniswami, M.: Internet of things (iot): A vision, architectural elements, and future directions. Future generation computer systems 29(7), 1645–1660 (2013)
14. Guillen, E., Sánchez, J., Paez, R.: Inefficiency of ids static anomaly detectors in real-world networks. Future Internet 7(2), 94–109 (2015)
15. Kulkarni, A., Sathe, S.: Healthcare applications of the internet of things: A review. International Journal of Computer Science and Information Technologies 5(5), 6229–6232 (2014)
16. Kwiatkowska, M., Norman, G., Parker, D.: Markov decision process, http://www.prismmodelchecker.org/lectures/biss07/04-mdps.pdf
17. Kwiatkowska, M., Norman, G., Parker, D.: Prism: Probabilistic symbolic model checker. In: International Conference on Modelling Techniques and Tools for Computer Performance Evaluation. pp. 200–204. Springer (2002)
18. Liang, L., Zheng, K., Sheng, Q., Huang, X.: A denial of service attack method for an iot system. In: Information Technology in Medicine and Education (ITME), 2016 8th International Conference on. pp. 360–364. IEEE (2016)
19. Long, N., Thomas, R.: Trends in denial of service attack technology. CERT Coordination Center (2001)
20. Mahoney, M.V., Chan, P.K.: Learning nonstationary models of normal network traffic for detecting novel attacks. In: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 376–385. ACM (2002)
21. Mell, P., Hu, V., Lippmann, R., Haines, J., Zissman, M.: An overview of issues in testing intrusion detection systems (2003)
22. Mirkovic, J., Dietrich, S., Dittrich, D., Reiher, P.: Internet Denial of Service: Attack and Defense Mechanisms (Radia Perlman Computer Networking and Security). Prentice Hall PTR, Upper Saddle River, NJ, USA (2004)
23. Mukkamala, S., Janoski, G., Sung, A.: Intrusion detection using neural networks and support vector machines. In: Neural Networks, 2002. IJCNN'02. Proceedings of the 2002 International Joint Conference on. vol. 2, pp. 1702–1707. IEEE (2002)
24. Parno, B., Perrig, A.: Challenges in securing vehicular networks. In: Workshop on hot topics in networks (HotNets-IV). pp. 1–6. Maryland, USA (2005)
25. Roesch, M., et al.: Snort: Lightweight intrusion detection for networks. In: Lisa. vol. 99, pp. 229–238 (1999)
26. Roman, R., Zhou, J., Lopez, J.: On the features and challenges of security and privacy in distributed internet of things. Computer Networks 57(10), 2266–2279 (2013)
27. Safavian, S.R., Landgrebe, D.: A survey of decision tree classifier methodology. IEEE transactions on systems, man, and cybernetics 21(3), 660–674 (1991)
28. Suo, H., Wan, J., Zou, C., Liu, J.: Security in the internet of things: a review. In: Computer Science and Electronics Engineering (ICCSEE), 2012 international conference on. vol. 3, pp. 648–651. IEEE (2012)
29. Talpade, R., Madhani, S., Mouchtaris, P., Wong, L.: Mitigating denial of service attacks (Jan 29 2003), uS Patent App. 10/353,527
30. Vattakunnel, A.J., Kumar, N.S., Kumar, G.S.: Modelling and verification of coap over routing layer using spin model checker. Procedia Computer Science 93, 299–308 (2016)
31. Zhang, G.P.: Neural networks for classification: a survey. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 30(4), 451–462 (2000)