SPATIO: end-uSer Protection Against ioT IntrusiOns

Gil Mouta, Miguel L. Pardal, João Bota, Miguel Correia
Instituto Superior Técnico, Universidade de Lisboa, Portugal

***Abstract*—The *Internet of Things* (IoT) is an emerging technology field where large numbers of physical objects communicate between themselves using Internet technology. IoT solutions are very diverse, ranging from simple toys to industrial applications. There are currently billions of IoT devices connected to the Internet, and this number has been growing exponentially in the recent years. The large amount of data being generated from the many devices in an IoT network makes it difficult to collect and analyse all the data. However, with this growth there also comes a growing security concern. With the use of IoT devices in the industrial and healthcare sectors, for example, a security incident can have far reaching consequences in the real world. It is imperative to detect attacks as fast as possible, in time to prevent significant damage.**

The continuous flow of data may be handled with a stream processing approach, a data processing paradigm in which high-rate data sources are processed and generate results on-the-fly. Based on this approach, we propose SPATIO (end-uSer Protection Against ioT IntrusiOns), an anomaly detection system designed for the IoT using machine learning to discover and alert on anomalies happening in an IoT network but takes a *fog computing* approach by using devices on the IoT network, such as routers, to collect and transform network traffic into flow metrics. Doing this transformation closer to the edge reduces the bandwidth cost on the network and allows anonymization of data before being sent outside the network, to the cloud or a server running *outlier detection* algorithms to generate timely alerts of network anomalies. We evaluate SPATIO by developing a prototype testing it on an existing public dataset of IoT attacks. We measured the *accuracy* of the machine learning approach, reaching close to 80% detection rate in the best scenario, and compared the *performance* of offloading work to gateway devices in the IoT network versus a centralized approach, in which the fog approach shows advantages in both network load as well as attack detection latency.

I. Introduction

The Internet of Things (IoT) is one of the fastest growing paradigms today. It consists of many objects of daily use with embedded smart sensors and computational resources, connected to the Internet. In 2018, there were 7×10^9 connected IoT devices [1], and it is estimated that the number of IoT devices will surpass the number of non-IoT active device connections by 2020. With IoT becoming more pervasive, it becomes a target for malicious agents. According to NETSCOUT's Threat Intelligence Report [2], IoT devices are attacked within 5 minutes of being plugged in, and targeted by specific exploits in 24 hours, and a survey by the Ponemon Insitute [3] in 2018 reveals that 21% of respondent organizations had a cyber attack or data breach caused by an unsecured IoT device in 2017.

IoT devices often present certain characteristics, such as reduced dimensions, low energy consumption, limited network (Internet) connection, limited processing and storage resources and low cost. These characteristics help disseminate the IoT paradigm to consumers, but they also make traditional security tools less effective. For example, traditional cryptography is harder to apply in IoT devices, due to the heavy use of resources that such algorithms put on the device [4]. Another aggravating factor in IoT security is the quantity of devices, many of which are of different types, from sensors to actuators, and from different vendors, highly increasing the chance that at least one of these devices will have a vulnerability or be misconfigured. Even if the IoT device being targeted is not the target desired by the attacker, a compromised device can act as a foothold on a network and allow it to launch more attacks.

Intrusion Detection Systems (IDS) are a common component of network security systems. They help discover, determine and identify unauthorized use, duplication, alteration and destruction of information and information systems [5]. IDS can monitor activity on the network, or run on individual hosts or devices, using data collected locally about events taking place, such as activity logs and system calls. They can also employ different detection methods, such as recognizing known attacks, or measuring the normal behavior of a device, and labeling any behavior differing from it as anomalous.

While IDS are one of the primary tools used for protection of traditional networks and information systems, some characteristics of the IoT make traditional IDS solutions inadequate [6]. The high number of small devices with constricted resources make host-based detection harder, while the high volume of traffic generated by these devices, on a high bandwidth network running on technology such as the recent 5G cellular network, make near real-time network level analysis more difficult. It is also important that modern systems be able to deal with the growing concern about data privacy.

The *fog computing* paradigm evolved to deal with some of these issues, where focus is shifted away from a centralized cloud computing environment towards the edge of the network. The main idea behind this paradigm is that gateway devices such as routers or base stations in an IoT network utilize their processing and storage resources to process data closer to its source, reducing latency and processing cost.

A. Objectives

In this dissertation, we propose an architecture to deal with the challenges of designing an IDS for the IoT, namely the limited characteristics of the IoT devices, the high amount of data generated by the large amount of devices, most of which are sensors, and the need for data privacy. We compare different types of machine learning approaches to intrusion detection, and analyze the advantages of offloading work closer to the edge of the network.

To this end, we developed a prototype of our architecture, SPATIO (end-uSer Protection Against ioT IntrusiOns), which uses outlier detection algorithms to detect anomalies in IoT networks. The core machine learning model works in a streaming paradigm, analyzing data as it comes and producing alerts in a timely manner.

We also compared a fog computing approach versus a more centralized one, by distributing some processing of data to the edge of the network, where gateway devices monitor the IoT device network traffic, and transform it into flow metrics which are consumed by the machine learning algorithm.

In developing SPATIO, we analyzed available machine learning frameworks and discuss which is best for our case. We also compare several public datasets on IoT intrusions.

We evaluate our prototype by measuring its accuracy in detecting intrusions, as well as counting the number of false alarms produced. We also evaluate the advantages of offloading work to devices in the edge of the network, by comparing its performance to a centralized approach. We do this evaluation on a public dataset of IoT intrusions.

B. Context

Telecommunications networks play an important role: they connect the smart objects to the Internet, and are interested in deploying IoT networks in industrial contexts, or for large cultural events such as concerts. As such, it is in their interest to protect clients from potential network intrusions that can cause damage to their clients or their brand. We developed this dissertation in collaboration with Vodafone Portugal, a big portuguese telecom company and internet service provider, to better understand the current trends in IoT networks, such as what types of devices or architectures are being used.

II. Background and Related Work

We start by presenting the Internet of Things (IoT), discuss the fog computing paradigm and review research by other authors in Intrusion Detection Systems for the IoT.

A. Internet of Things

The IoT is the ever-growing network of physical objects that are connected to the Internet. It consists of many objects of daily use with embedded smart sensors and computational resources. These objects make possible new and innovative applications [7].

According to an IoT Analytics research [1], in 2018 there were 7 billion connected IoT devices, and by 2020 it is expected that the number of IoT devices will surpass the number of non-IoT active device connections worldwide.

With the ubiquity of the IoT, it makes sense to separate an industrial focused use case from a consumer use case.

The Industrial Internet of Things (IIoT) is the concept of IoT applied to the industry. For example, IoT has been applied to the mining industry, for the *safety of miners*. Environmental sensors are deployed to detect flooding, fires, gas or dust explosions, cave-ins, toxic gas and other risk factors. They also allow precise location and identification of miners. By using wireless communication technology, like WiFi and RFID (Radio-frequency Identification) to send the collected information to the surface, mining companies can keep track of the miners and analyze critical safety data to enhance safety measures [8].

On the consumer side of the IoT, *smart home* technology uses sensors and actuators in common household objects to provide a better quality of living. Home parts are equipped with different technologies for more intelligent monitoring and remote control, as well as allowing interaction between each smart component, such that everyday house activites are automated without user intervention, or with more convenient remote control [9]. Some of its applications include lighting, air conditioning, energy management, security and control of other home appliances. For example, a kitchen can have refrigerators, microwaves, coffee makers and dishwashers with IoT capabilities [10]. One example is the Internet Refrigerator, as developed by LG Electronics, an Internet enabled refrigerator which lets the users download recipes and display them on a liquid-crystal display (LCD) screen. The refrigerator is also able to keep track of its inventory automatically, and alert the user to what is there and what is missing.

B. Fog Computing

Fog computing is a distributed computing paradigm that acts as an intermediate layer between services provided by the cloud and the edge of the network, where the IoT devices are [11]. The concept was introduced by Cisco in order to address the challenges of IoT applications in conventional cloud computing, such as the high amount of nodes, with a wide geo-distribution, making a traditional geographically centralized cloud solution not ideal, especially when dealing with latency-sensitive operations.

A fog computing environment typically consists of gateway devices, such as routers and base stations, placed in proximity to the IoT devices and sensors. As opposed to the more resource constrained IoT devices, these intermediary devices have higher processing, storage and networking resources to be able to do some pre-processing of the data coming from the IoT devices before being sent to the cloud.

A fog computing approach can bring the following advantages [12]: Reduction of network traffic by doing pre-processing of data before being sent to the cloud, better serve local events by using locally stored information to act on requests, lower latency by doing processing closer to source, instead of having to do a round-trip to the cloud and back,

better scalability in the cloud by reducing the amount of work and better control of data privacy, as data is processed closer to it's owner.

These advantages make fog computing a better fit for IoT applications when compared to a traditional centralized approach.

An example of a fog computing application is FogLearn [13], a fog architecture framework which, amongst other scenarios, was used in detecting diabetes patients. It leverages data collected from smart wearables in patients, using fog devices to reduce latency and increase throughput. Data collected from patients in the edge layer is first pre-processed in the fog layer, before being sent to the cloud layer for further analysis and long term storage.

C. Machine Learning for Intrusion Detection

Security is a growing concern in the IoT. IoT inherits the traditional network security problems, and adds its own set of problems. For example, IoT devices tend to have limited computational resources, and a limited battery life. This makes conventional cryptography harder to apply to such devices, due to to the heavy use of resources such algorithms put on the device [4].

Intrusion Detection Systems (IDS) are a common component of network security systems. They help discover, determine and identify unauthorized use, duplication, alteration and destruction of information and information systems [5].

When dealing with anomaly detection, for the purpose of network intrusion detection, a fast solution with as little human effort required as possible is preferred, to ensure timely detection of any possible attack. This is where machine learning solutions come in.

Machine learning consists of automated detection of meaningful patterns in data [14]. It is an emerging field of data analysis which became common standard in solving most problems that require analyzing a large amount of data, finding complex patterns that a human programmer cannot explicitly define.

In this section, we look at some of the machine learning solutions for anomaly detection proposed by other authors.

Gonçalves et al. [15] propose a machine learning and data mining approach for analysing log data and semi-automatically discovering misbehaving hosts, without having to instruct the system about how hosts misbehave. A probability based clustering algorithm, the Expectation-Maximization algorithm, is used, which has the benefit of not requiring prior knowledge of the distribution of each feature and other parameters, making it adequate for clustering large datasets. Classification algorithms, a method of supervised machine learning, are then used to assign clusters to classes, such as "normal" and "abnormal". This is the phase that requires the most human effort, but after the creation of the classification model, human intervention is only needed if the classification results are wrong, to update the model. The authors concluded that while the presented process did extract relevant security information from the logs that otherwise is not directly observable, it is not accurate enough to take automatic actions to deal with the misbehaving host.

Another example of machine learning applied to IoT intrusion detection, Meidan et al. [16] propose a network-based approach to anomaly detection using autoencoders to detect botnets. The use of autoencoders has the advantage of not needing manual classification of what is malicious or benign, meaning unknown attacks can be detected. Also, by profiling each device with a separate auto encoder, this method addresses the growing heterogeneity of IoT devices. For experimental evaluation, the authors created and used a dataset out of devices purposefully compromised to the Mirai and BASHLITE botnets. They were successful in detecting every attack launched by each compromised IoT device, i.e., true positive rate of 100%. Also, compared to other algorithms commonly used in anomaly detection (Local Outlier Factor, One-Class Support Vector Machine and Isolation Forest), this method got the fewest false alarms, while also being the fastest method at detecting the attacks.

Chawathe [17] attempted to achieve similar accuracy, and potentially better performance, than the system proposed by Meidan et al. [16]. The author proposes a network-based method for monitoring botnets on IoT devices, requiring no access to the device other than the ability to monitor network traffic from some other host on the network. The author suggests the use of a simpler method based on supervised machine learning through the use of classifiers alone. The author concludes that simple classification methods can achieve high accuracy in classifying network activity into the 11 classes, as well as achieving those high accuracy values by using only a small fraction of the original attributes (over 99% accuracy using only 20 of the 115 attributes on one of the devices in the dataset).

Outlier detection algorithms work by finding values which deviates from other observations on data. This type of algorithm lends itself to intrusion detection, since an attack is by definition an outlier, as it deviates from the normal behavior of a system. Auskalnis et al. [18] proposes the usage of the Local Outlier Factor (LOF) algorithm to detect anomalies in a computer network. The LOF algorithm detects outliers by evaluating each event's uniqueness based on the distance from the k-nearest neighbours. The expectation is that the density of events around an outlier is significantly different than the density of its neighbours. The authors evaluate their solution on an existing dataset, the NSL-KDD dataset[1]. An accuracy rate of 0.84 was obtained. The results are also compared to the accuracy rates of classification methods, and obtain the same values for accuracy, however the authors conclude using LOF or other unsupervised outlier detection methods have the advantage of not needing labelled

[1] https://www.unb.ca/cic/datasets/nsl.html

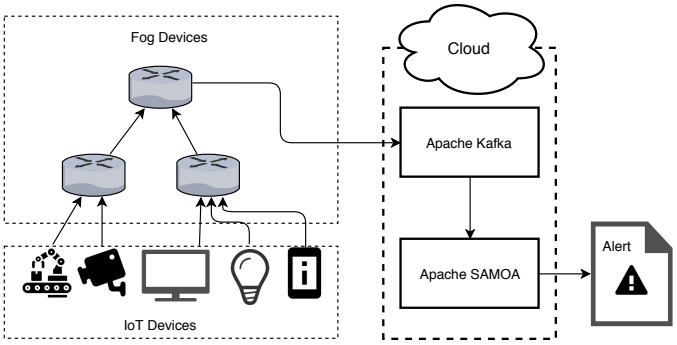

Fig. 1: Overview of the SPATIO architecture

datasets for learning.

III. SPATIO

In this chapter we present the SPATIO (end-uSer Protection Against ioT IntrusiOns) system and its prototype implementation. We start with an overview of the architecture and data pipeline in Section III-A, and then a more detailed explanation of method and tool selection for each step, starting with the machine learning framework in Section III-B, feature extraction in Section III-C and feature selection in Section III-D. We conclude with a summary of the SPATIO system.

A. Architecture

The goal of SPATIO is to detect anomalies in an IoT network. For this we analyze network traffic, as the heterogeneity of IoT devices, with varied characteristics, capacities and goals, makes host-based detection difficult. We also expected to obtain network captures from our collaboration with a telecom company which deploys IoT networks. Figure 1 gives an overview of the SPATIO architecture.

Devices in the *fog* of the network, such as routers, collect incoming and outgoing network traffic of the IoT devices (left side of Figure 1). We then take a fog computing approach, as described in Section II-B, by having these devices also process the raw data traffic into flow features. This process is described in Section III-C. By doing this transformation closer to the data source, we try to reduce network traffic, as only the reduced flow features need to be sent to the cloud, instead of the entire raw traffic. We also reduce the work needed to be done in the cloud, increasing scalability.

Another advantage of the fog computing approach is increased data privacy. By summarizing the whole traffic into flow features, the actual content of communication is not sent. Additionally, information identifying the devices, such as the IP address, can be transformed into an alias to identify the device.

Flow features are sent to the machine learning algorithm through Apache Kafka[2]. Apache Kafka is a distributed streaming platform which works like a Publish and Subscribe system, acting as a message queue. We choose Kafka for its ability to be distributed, preventing a single point of failure, as well as its interoperability with our chosen machine learning framework, Apache SAMOA, described in Section III-B.

SPATIO's end result are *alerts*. These alerts indicate an anomaly in the network, affecting a certain device. This anomaly might or might not be the result of an ongoing attack. Human validation is required to differentiate a benign anomaly from an intrusion. This validation can later be done to teach a classification algorithm to be able to differentiate benign from malicious anomalies, although that was not explored in this research.

B. Streaming and Machine Learning Framework

When choosing a machine learning framework, we looked at available data analysis frameworks which featured machine learning or integrated with a framework that did.

Streaming data analysis has the advantage of not requiring data to be stored in memory, and providing quicker results. In an IoT environment, the large number of devices can generate a stream of data that is not feasible to be stored, and dealing with intrusion detection requires timely detection, which makes a streaming approach a good fit. As such we took a streaming approach for SPATIO, using Apache SAMOA.

Apache SAMOA and Apache MOA[3] are open source frameworks for data stream mining written in the Java programming language. They both function very similarly in terms of programming interface. However, SAMOA was built to be used in a distributed environment. To this end it can integrate with Apache Storm, Apache S4 or Apache Samza, other stream processing frameworks which do not support machine learning natively, but are made to be distributed over several nodes. Although in our prototype for evaluation we do not use this feature, opting for Apache SAMOA's local mode, in a real world scenario having the IDS be distributed is advantageous as it prevent a single point of failure and spreads computation allowing for faster response times.

SAMOA is an Apache Incubator project, meaning it is still a developing project not ready to enter the Apache Software Foundation. This reflects on the fact that it lacks diversity in available machine learning algorithms, and a somewhat lacking documentation. To solve the former problem of lack of machine learning algorithms, we used MOA's algorithms in SAMOA. MOA is SAMOA's centralized counterpart. It is a more mature project, leading it to have more machine learning algorithms available, in several categories from basic classification and clustering tasks to outlier detection and active learning. To export MOA's algorithms to SAMOA, we used an existing library[4], although this library had not been updated in 5 years and required some modifications before it could be used.

We use outlier detection algorithms, which are a good fit for intrusion detection since by definition, an attack is an anomaly.

[2]https://kafka.apache.org/

[3]https://moa.cms.waikato.ac.nz/

[4]https://github.com/samoa-moa/samoa-moa

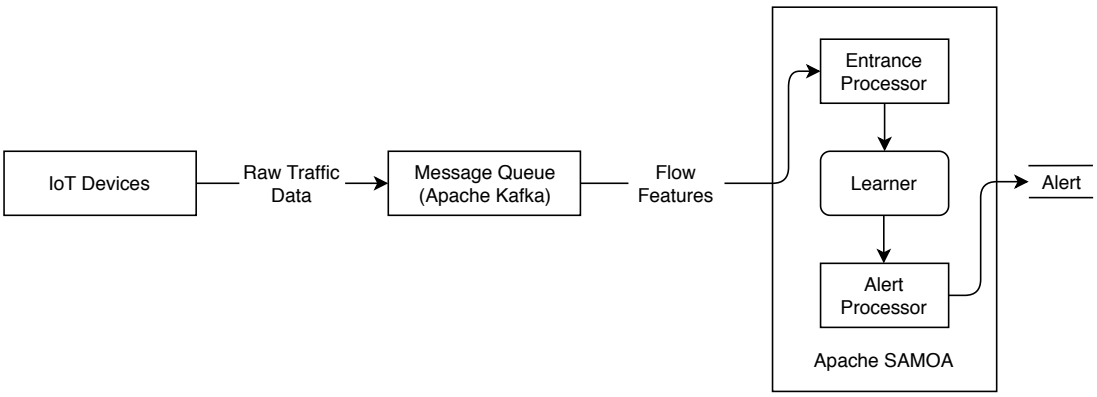

Fig. 2: Overview of the SAMOA topology used for SPATIO.

They are an unsupervised algorithm, and do not require a training period. SAMOA and MOA include the following outlier detection algorithms, which we test in Section IV-B2: ExactSTORM, ApproxSTORM, AbstractC and MCOD.

C. Feature Extraction

SPATIO monitors the network traffic in order to detect anomalies. To this end gateways, such as routers, to which the IoT devices are connected, are constantly capturing network traffic using tools such as tcpdump[5]. While most of these tools allow for filtering packets by port, address or protocol, or more detailed filters, for SPATIO we do not use this filtering feature and capture every packet arriving at the network card.

For SPATIO, feature extraction is done in IoT devices on the fog layer. Since it is important for an attack to be detected as quickly as possible, and these types of device can be resource constrained, we focus on analyzing the header of the packages, instead of analyzing the contents of each packet. Packet header anomaly detection also has the added advantage of remaining valid when the payload is encrypted, such as in an SSL session.

For our approach, we use single connection derived features, features derived from a single connection, or flow. A flow consists of all traffic with similar characteristics, usually source and destination address and ports.

To transform a series of network packets, which is what we are collecting with tcpdump, into flow records, we chose the NetMate-flowcalc[6] tool. NetMate (Network Measurement and Accounting Meter) is a network measurement tool used to generate statistics of network traffic. We chose NetMate for its ability to operate in real time, which is necessary for our streaming approach, for its ease of configurability, where each configuration, such as features extracted and the timeout period for flow creation, is easily changed in a rule file, and because it by default outputs the features in a comma separated value (csv) format which makes no further parsing required before sending the flow data to the machine learning algorithm.

NetMate captures 38 features by default, plus the [source address, source port, destination address, destination port] tuple used to identify the flow, i.e. all traffic which matches these same characteristics are grouped into a single flow, as long as the time between packets does not cross a certain threshold, in which case a new flow is created.. We use those features as the base features for SPATIO, before feature selection described in the next section.

D. Feature Selection

Feature selection consists of selecting a subset of features in order to reduce the dimension of the problem. This technique brings several advantages, such as a better generalization of the machine learning model to the problem, reducing overfitting; reduction of network bandwidth used on sending the flow features to the nodes running the machine learning model; avoiding the *curse of dimensionality*, in which more features makes the algorithm less efficient [19]. For anomaly detection, the more features we consider in a flow, the more likely flows are to be different from each other, making outlier detection more difficult. As such, we can consider feature selection to be the process of identifying and removing as much irrelevant and redundant information as possible.

For feature selection we use WEKA[7]. WEKA is a suite of machine learning tools and algorithms written in Java. It contains several feature selection algorithms. For our purposes, we need a feature selection algorithm which runs on unlabeled data. We chose to use Principal Component Analysis as a feature evaluator. Principal Component Analysis (PCA) works by transforming the current set of features into a new reduced set of features, known as the Principal Components. This new set is choosen such that each component is orthogonal to each other, i.e. uncorrelated. Two features are correlated when a change in one leads to a change in the other. An example of two correlated features are total_fpackets (total packets in the forward direction) and total_fvolume (total bytes in the forward direction), since one can expect the number of bytes

[5]https://www.tcpdump.org/

[6]https://github.com/DanielArndt/netmate-flowcalc

[7]https://www.cs.waikato.ac.nz/ml/weka/

to increase with more packets sent. This does not always need to be observed for the features to be correlated, but the more present it is, the more strongly correlated the features are.

The new set of features produced by PCA are essentially linear combinations of the original features, where each feature is given a weight. Although new set of features can be used for machine learning directly, we thought it would be interesting to analyse which features of the original set extracted would be more relevant to intrusion detection in our dataset. To this end, we summed the weights given to each feature in each Principal Component, considering that the higher this sum, the more relevant the feature is.

E. Summary

This chapter presented SPATIO, a system to detect anomalies in an IoT network. Figure 3 summarizes the data-flow between the components of the SPATIO architecture. Network traffic is collected from IoT. This traffic get processed into flows through feature extraction. The system can then use these flows, or further refine them by going through feature selection. Feature selection analyses the extracted flows, and through correlation analysis returns the n-sized subset of features that more accurately represent the dataset. This process only has to be done once. After feature extraction, the flows are sent to the machine learning framework, running an outlier detection method. This outputs alarms, which indicate an anomaly in a certain device.

IV. Evaluation

In this chapter we present the experiments that were done to evaluate the SPATIO prototype using a public dataset. The experiments assessed the accuracy of the prototype and the choice of architecture. We were interested in answering the following questions about our system:

- What percentage of attacks can our system detect?
- How many false alarms will be generated?
- What features are relevant for intrusion detection systems, and how does the quantity of considered features affect the results?
- What are the advantages of offloading work to fog devices?

A. Dataset

To evaluate SPATIO, we tested our prototype on an existing dataset. We were originally planning on capturing our own set of data from a real IoT network in collaboration with a telecom company, but we were not able to find a proper IoT environment currently running and get the required authorizations for data capture in time. Since we are interested in testing the feature extraction from the raw data traffic, we look exclusively at datasets that provided the entire network capture, in a pcap format. The datasets we considered are summarized in Table 4.1.

To choose a dataset, we focused on certain qualities: the dataset had to be labeled, so we could easily test the accuracy of our solution, it had to be IoT related, and have as many devices as possible. We decided on using a dataset collected for ACM SOSR 2019 by Hamza et al. [20]. Although this dataset cover less devices and less attack types, focusing only on Denial of Service (DoS) attacks, it was the IoT dataset with the best labeling for our purposes.

The chosen dataset consists of network traffic collected in a lab. The lab is comprised of a TPLink gateway with OpenWrt firmware serving 27 IoT devices, of which 10 are used as victims for attacks. The network traffic is captured at the gateway. Benign traffic is collected by simulating possible user interaction with each IoT device, while malicious traffic is captured by running a script exploiting known vulnerabilities to carry out both reflected and direct DoS attacks. Traffic is captured over a month, with two periods of around a week where attacks happen, which we name Period 1 and Period 2. The attacks present in the dataset consist of direct DoS attacks, in which the targeted devices are the victims, and reflected DoS attacks, in which the targeted device is used to attack another outside device. All of the attacks are sustained for 10 minutes, at various packet rates, and in total there are 200 attacks.

B. Accuracy Evaluation

The first set of tests measures the accuracy of the system. We take into consideration the following metrics:

- *Total alarms*, the total number of alarms generated.
- *Attacks detected*, the number of attacks detected. To calculate this metric, we consider any attack during which an alarm is generated, linked to the device being attacked, as being detected. Multiple alarms during the same attack do not further increase this number.
- *Percentage of Attacks detected*, the percentage of attacks detected, calculated by dividing *Attacks detected* by the total number of attacks as stated by the ground truth of the dataset.
- *Alarms during attack*, the number of alarms generated during attacks. This includes multiple alarms generated in a single attack.
- *Percentage of Alarms during attack*, the percentage of alarms which fall inside an attack window for that device. This is calculated by dividing *Alarms during attack* by *Total alarms*. The remaining percentage is the false alarm percentage.
- *False Alarms (ALL)*, the number of false alarms. A false alarm is an alarm which happens while there is no attack occurring on the related device.
- *False Alarms (1min)*. This metric works like the one above, but after an alarm occurs, no more alarms can occur for one minute.
- *False Alarms per day*, the amount of false alarms generated per day, based on the results of the metric *False Alarms (1min)*.

1) Evaluation Environment: Since these tests do not consider the performance of the solution, and only the accuracy of the machine learning model, they were run on a dedicated machine with the pipeline simplified to SAMOA reading the

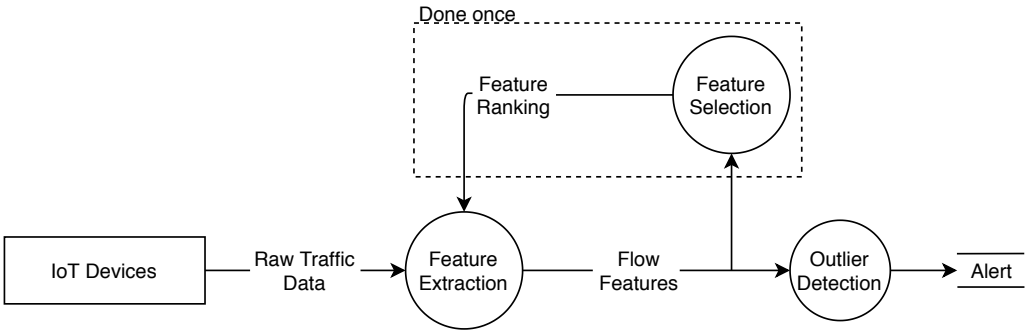

Fig. 3: Data-flow diagram for the SPATIO architecture.

previously generated flow metrics directly from a file. This machine has 16 GB of RAM, 1 TB of disk storage and is running Debian GNU/Linux Buster.

After collecting the alerts in a file, a Python script is run which compares them with the ground truth provided by the dataset, providing the metrics described in the previous section.

2) Outlier Detection Algorithms: We start by measuring how the considered metrics vary by using different outlier detection methods. The results are shown in Tables I and II. We defer the discussion of these results to Section IV-B4.

TABLE I: Evaluation of the accuracy metrics according to tested outlier detection algorithm for Period 1.

	Period 1			
Algorithm	MCOD	ExactSTORM	ApproxSTORM	AbstractC
Total Alarms	218203	222147	218912	219045
Attacks Detected	43	43	43	43
%Attacks Detected	79.63%	79.63%	79.63%	79.63%
Alarms during attack	5411	5459	5441	5438
%Alarms during attack	2.48%	2.46%	2.49%	2.48%
False Alarms (all)	212792	217238	213334	213539
False Alarms (1min)	6759	6759	6759	6759
False Alarms per day	563	563	563	563

TABLE II: Evaluation of the accuracy metrics according to tested outlier detection algorithm for Period 2.

	Period 2			
Algorithm	MCOD	ExactSTORM	ApproxSTORM	AbstractC
Total Alarms	130312	97216	95841	95443
Attacks Detected	61	58	58	58
%Attacks Detected	75.31%	71.60%	71.60%	71.60%
Alarms during attack	22681	20341	20228	20203
%Alarms during attack	17.41%	20.92%	21.11%	21.17%
False Alarms (all)	107631	97216	95842	95443
False Alarms (1min)	7999	5957	5953	5950
False Alarms per day	667	496	496	496

3) Feature Selection: As explained in Section III-D we use Principal Component Analysis (PCA) in WEKA to rank our features, by considering the sum of the weights of each Principal Component. We take one day of Period 1 where attacks happen as the training dataset to run PCA. Selecting a day with attacks ensures we capture the features more relevant to detect attacks, while using only one day ensures we do not overfit our selection.

With the feature ranking in mind, we tested how the number of considered features affect the metrics described in Section IV-B. We run five tests for each period, considering the 6, 12, 18, 24 and 30 most relevant features. We also compare with the base 36 feature result obtained in the previous section.

These tests use the same environment as the ones done in the previous section, and run the MCOD outlier detection algorithm. The results are shown in Tables III and IV.

TABLE III: Evaluation of the accuracy metrics according to changing number of features for Period 1.

	Period 1					
Number of Features	6	12	18	24	30	36
Total Alarms	50999	204649	204819	211459	218221	218203
Attacks Detected	38	39	39	43	43	43
%Attacks Detected	70.37%	72.22%	72.22%	79.63%	79.63%	79.63%
Alarms during attack	2069	4640	4686	4717	5413	5411
%Alarms during attack	4.06%	2.27%	2.29%	2.23%	2.48%	2.48%
False Alarms (all)	40042	204814	204477	212563	212792	212792
False Alarms (1min)	10115	5549	5186	6759	6759	6759
False Alarms per day	920	504	471	614	614	614

TABLE IV: Evaluation of the accuracy metrics according to changing number of features for Period 2.

	Period 2					
Number of Features	6	12	18	24	30	36
Total Alarms	22967	28423	82541	90330	92554	130312
Attacks Detected	53	55	56	58	58	61
%Attacks Detected	65.43%	67.90%	69.14%	71.60%	71.60%	75.31%
Alarms during attack	17001	18537	19020	19302	20663	22681
%Alarms during attack	74.02%	65.22%	23.04%	21.37%	22.33%	17.41%
False Alarms (all)	21142	26164	75983	78334	80444	107631
False Alarms (1min)	6079	6001	5931	6070	5957	7999
False Alarms per day	553	546	539	552	542	727

4) Discussion: We can see all the tested outlier detection algorithms give similar results. In Period 1, we were able to detect 79.62% of the attacks in all cases. However, a difference can be seen when comparing false alarms. The MCOD algorithm had the lowest amount of generated alarms, and one of the highest percentage of alarms that were generated

during an attack, leading it to have the least amount of false alarms. In a real life situation, a false alarm requires human intervention, wasting time. It also potentially leads to *alert fatigue*, a situation in which so many false alerts are being generated that the human operator becomes desensitized to them, leading to a longer response time to real attacks, or even missing them completely [21]. As such, a lower number of false alarms, and of alarms in general, is desirable. To this extent, we measured the number of false alarms generated if we blocked alarm generation for one minute after an alarm. The principle behind this idea is that when an anomaly occurs, whether it is benign or not, it can have an impact on multiple connections, generating several alerts in a short amount of time. This does seem to be the case, as this change reduces the number of false alarms considerably.. A similar idea can be applied to real attack alarms, to reduce the number of alarms in general. In Period 2, we observe a smaller percentage of detected attacks, and a smaller percentage of total alarms. This can occur because of the different devices being tested in the different periods, where attacks on this new set of devices are harder to distinguish from normal benign traffic.

In Section IV-B3 we tested how the considered metrics change as the number of features change. A noticeable pattern is that as less features are considered, less alarms are generated, but the number of attacks detected also lowers. This can be explained by the loss of information when removing features. Outlier flows could be outliers by varying a lot on a certain feature, which could then be removed in feature selection, thus this outlier is no longer classified as such. We can see that when we use only 6 features, the metrics of false positives start to get worse due to too much loss of information. It's also interesting to note that the number of false alarms tends to reduce as less features are used, but only because the total alarms is also reducing, as the percentage of false alarms actually increases with less features. Still, using a lower number of features is something to be considered, because although the detection rate of the system is lower, the bandwidth used to transmit the flows will also be lower, and less alarms in total will be generated, which can be useful in a situation where dealing with alarms has some resource cost associated. In Period 2, we see a bigger drop off in accuracy with less metrics. This can be caused by the fact that feature selection was done over the dataset of Period 1, making those features more accurate at representing traffic from Period 1.

C. Architecture Comparison

The second set of tests compares the chosen architecture of SPATIO, where work is offloaded to fog devices, with a centralized approach, i.e. an architecture that skips this step, choosing instead to transmit the captured packets directly to a central server running the machine learning algorithm, which then also becomes responsible for feature extraction.

1) Evaluation Environment: To test our architecture in comparison with a centralized approach, we set up two scenarios using virtual machines. In the first scenario, testing the fog computing approach, two VMs act as gateway devices, collecting the IoT traffic and running the feature extraction, and then sending the flow features to a VM representing the cloud, running Apache Kafka and Apache SAMOA. In the second scenario, testing the centralized approach, the two gateway VMs still collect traffic from the IoT devices, but this traffic is sent directly to the cloud VM, with no feature extraction. The cloud VM is then responsible for doing feature extraction.

The gateway VMs were modelled after specifications of gateways being used in real IoT networks, given to us by a telecommunications company. As such, it contains 512MB of RAM, and 4GB disk storage. The cloud VM has 8GB RAM and 32GB disk storage. Both VMs run Debian GNU/Linux Jessie 64 bit.

To emulate the IoT device traffic, tcpreplay[8] was used to replay the dataset in a dummy network interface.

2) Network Bandwidth Usage: In this section we consider the differences of network bandwidth usage between sending the raw traffic in the network, or only sending the extracted flow records. We measure the size of the raw traffic for each period, and the size of the flow records. We then calculate the average bandwidth in bps by dividing the total size by the duration of the experiment. The results are shown in Table V.

TABLE V: Network bandwidth difference between centralized and fog approach.

	Total size raw traffic	Total size flows	Bandwidth avg sending raw traffic	Bandiwdth avg sending flow features
Period 1	28.2 GB	1.72 GB	29209 bps	1787 bps
Period 2	25.6 GB	4.50 GB	28998 bps	5092 bps

We can see using flow metrics results in much lower bandwidth requirements, reaching a 90% reduction in the case of Period 1.

3) Latency: In this section, we compare the different architectures by measuring the latency in anomaly detection. To do this, we use 10 benchmark alerts, and measure how fast each solution creates that alarm. We repeat each test five times, and take the average as our value. Since the dataset contains traffic captured over a month, o reduce the time it takes to perform the test and to emulate more network traffic, one day was chosen from the dataset and data was replayed at a rate of 100 MB/s. The results are shown in Figure 4, where each data point represents a benchmark anomaly.

[8]https://tcpreplay.appneta.com/

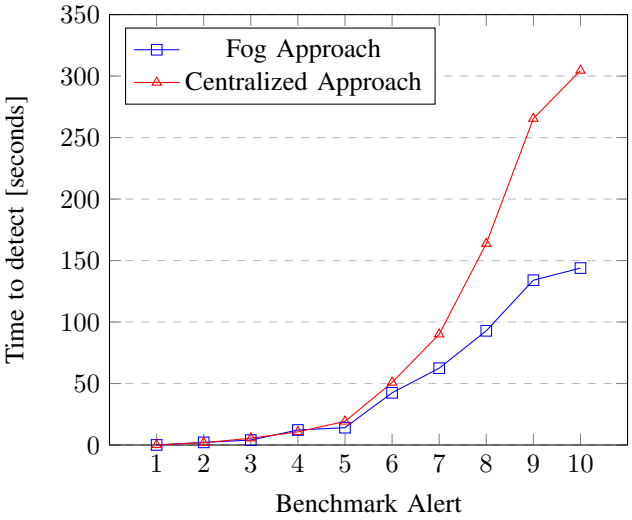

Fig. 4: Detection latency in fog versus centralized approach

4) Discussion: IoT networks tend to have many connected devices, and use lower bandwidth technologies such as Low-power wide-area networks (LPWAN) for long-range communications at low power consumption and low data rate. As such, minimizing network traffic is advantageous. Transforming raw traffic into flow records can reduce network usage by a lot, as would be expected since an entire connection of packets is transformed into a single line of features. An interesting fact to note is that despite having lower traffic size, Period 2 flow records are bigger. This can be explained by the fact that a flow can cover an entire connection, regardless of number/size of packets. For example, if a device was to download a video from the internet sized 1GB, this entire 1 GB worth of traffic would be transformed in a single flow. In fact, by further analysing which devices are being used in Period 2, we can see devices such as Chromecast, used to stream video, and Amazon Echo, a virtual assistance with features such as playing music from the Internet.

In the latency test, we can observe both approaches get worse latency as time goes by. This indicates data is being backlogged somewhere. In the centralized approach, this latency increase is more accentuated, which could indicate feature extraction on the server accounts for a big portion of the backlog, showing the advantage of distributing this task to the gateway devices. It is important to note that this test is done under a constant rate of 100 MB/s of traffic. In a situation with lower data rates, the difference between the two approaches is likely to reduce.

D. Summary

The experimental results of the SPATIO prototype regarding accuracy and false positive rate allow us to reach a few conclusions. First, the choice of algorithm is not as important as the features selected to represent the data. We can observe that although not much changes in accuracy between algorithms, selecting different amounts of features lead to a big change. When developing an IDS, one must consider the trade off between having many features, leading to an higher network and computing cost, and having fewer features, leading to a lower accuracy. We can also observe how proper feature selection impacts the results. In our prototype, feature selection was done based on data from Period 1 alone. This leads to a situation where, when using a lower amount of features, the system performs better on Period 1 than on Period 2.

We also observed the advantages of the fog computing approach taken in SPATIO. We were able to reduce network bandwidth usage by 90% by preprocessing the data closer to the data source. We also reduced load on the cloud server, leading to a lower alarm latency. This means that alarms are delivered faster, which can bring benefits when responding to attacks.

V. Conclusion

In this dissertation, we analysed the IoT as a growing market, and identified the challenges it brings. We contribute with a solution for better security in IoT, as several reports show it to be a growing concern. Specifically, we address how to design an IDS for the IoT, having to deal with the IoT device's constrained processing and storage resources, the high amount of traffic in an IoT network, and the timely response time needed when dealing with potential security alerts.

Our presented solution is called SPATIO, a system to detect intrusions in the IoT. SPATIO finds intrusions in a network by monitoring traffic of the IoT devices, and feeding it to a machine learning model running outlier detection algorithms, which are a good fit to this problem as attacks are outliers by nature. To deal with the high amount of traffic in the network, while providing timely alerts, we adopted a streaming solution, which processes the incoming data flows in a continuous manner. This means we do not need to wait for a window of processing, as would be the case in batch processing, providing a faster generation of alerts. This streaming approach also has the advantage of not requiring the storage of data, which would not be practical in the IoT devices due to storage constraints, nor in the cloud, do to the high amount of devices generating high amounts of traffic.

We also take a fog computing approach in SPATIO. Feature extraction, i.e. generating useful metrics based of flows of data, is done on devices closer to the edge of the network. Using the IoT devices themselves, i.e. edge computing, would not be practical due to the constrained processing resources they possess. Instead, by using gateways such as routers and base stations, we reduce work load on the cloud (central servers), increasing scalability, while also reducing overall network traffic, since only the transformed flow features are sent, instead of the entire raw traffic. Additionally, fog computing allows for better data privacy control, as data is processed closer to the owner of the data. This allows the removal of sensitive data and any personally identifiable information before it is sent to the cloud.

A. Achievements

We developed a prototype of SPATIO and used it to run tests on a public IoT network intrusion dataset. In the best case, we

manage to detect 79.63% of the attacks, while producing on average 614 false alarms per day, over an eleven day period. While this amount of false alarms are somewhat high, we believe it can be much further reduced and discuss a possible way in Section V-B.

In our evaluation,we observed a trade off between having an higher amount of features, resulting in a better accuracy, and a lower amount of features, reducing network traffic and requiring less computational power.

We also observed the advantages of taking a fog computing approach. SPATIO reduces traffic in network by 90% by sending flow features to the cloud instead of the entire raw traffic. It also shows a clear advantage in timely processing of the information when offloading work to the fog devices, especially in a period of high constant throughput, where a backlog of data starts forming.

B. Future Work

While evaluating our prototype, it became clear it produced too many false alerts. This can lead to alert fatigue, a situation in which so many false alerts are being generated that the human operator becomes desensitized to them, leading to a longer response time to real attacks, or even missing them completely. To deal with this issue, a classification algorithm can be added to the SPATIO machine learning pipeline, after the outlier detection. The idea would be to further classify alerts into malicious and non malicious. While this would require initial human effort, the system would learn from each false alarm and gradually produce fewer and fewer false alarms.

Something that difficulted our work when evaluating SPATIO was the lack of a good IoT dataset containing traffic from a high number of devices in a real scenario. Although the lack of such public datasets might be understandable, by the fact that real traffic captures might contain private data and therefore not many entities currently running IoT networks would be willing to part with their captures. Nonetheless, SPATIO would benefit from a dataset containing more throughput and devices, mirroring a real life scenario.

Finally, as computational resources become cheaper, more work can be offloaded to fog devices. An interesting path for future work is to further research what can be offloaded to these devices. A possible approach would be using federated learning, as recently proposed by Google [22], where the fog devices would download the machine learning model from the cloud, update it with local data, and send back the update to the cloud, allowing distributed learning from the collected data, without the data ever leaving the devices.

REFERENCES

[1] IoT Analytics, "State of the IoT & Short term outlook 2018," 2018.

[2] NETSCOUT, "Dawn of the Terrorbit Era," 2018.

[3] Ponemon Institute, "Second Annual Study on The Internet of Things (IoT): A New Era of Third-Party Risk," 2018.

[4] W. Trappe, R. Howard, and R. S. Moore, "Low-Energy Security: Limits and Opportunities in the Internet of Things Low-Energy Security: Limits and Opportunities in the Internet of Things," *IEEE Security & Privacy*, 2015.

[5] S. Mukkamala, A. Sung, and A. Abraham, "Cyber Security Challenges: Designing Efficient Intrusion Detection Systems and Antivirus Tools," *Enhancing Computer Security with Smart Technology*, 2005.

[6] B. B. Zarpelão, R. S. Miani, C. T. Kawakani, and S. C. de Alvarenga, "A survey of intrusion detection in Internet of Things," *Journal of Network and Computer Applications*, 2017.

[7] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications," *IEE Communications Surveys & Tutorials, Volume 17*, 2015.

[8] W. Qiuping, Z. Shunbing, and D. Chunquan, "Study on key technologies of Internet of Things perceiving mine," *Procedia Engineering Vol. 26*, 2011.

[9] R. Kadam, P. Mahamuni, and Y. Parikh, "Smart Home System," *International Journal of Innovative Research in Advanced Engineering*, 2015.

[10] J. Barthold, "Changing the Way Houses Operate," 2005.

[11] R. Mahmud, R. Kotagiri, and R. Buyya, "Fog Computing: A Taxonomy, Survey and Future Directions," *Internet of Everything*, 2017.

[12] A. V. Dastjerdi, H. Gupta, R. N. Calheiros, S. K. Ghosh, and R. Buyya, "Fog Computing: Principles, Architectures, and Applications," *Internet of Things Chapter 4*, 2016.

[13] R. K. Barik, R. Priyadarshini, H. Dubey, V. Kumar, and K. Mankodiya, "FogLearn: Leveraging Fog-Based Machine Learning for Smart System Big Data Analytics," *International Journal of Fog Computing, Volume 1*, 2018.

[14] S. Shalev-Shwartz and S. Ben-David, *Understanding Machine Learning: From Theory to Algorithms*, C. U. Press, Ed. Cambridge University Press, 2014.

[15] D. Gonçalves, J. Bota, and M. Correia, "Big Data Analytics for Detecting Host Misbehavior in Large Logs," in *Proceedings of the 14th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, 2015.

[16] Y. Meidan, M. Bohadana, Y. Mathov, Y. Mirsky, D. Breitenbacher, A. Shabtai, and Y. Elovici, "N-BaIoT: Network-based Detection of IoT Botnet Attacks Using Deep Autoencoders," *IEEE Pervasive Computing*, 2018.

[17] S. Chawathe, "Monitoring IoT networks for botnet activity," *17th IEEE International Symposium on Network Computing and Applications*, 2018.

[18] J. Auskalnis, N. Paulauskas, and A. Baskys, "Application of Local Outlier Factor Algorithm to Detect Anomalies in Computer Network," *Elektronika ir Elektrotechnika Volume 24*, 2018.

[19] M. Verleysen and D. François, "The curse of dimensionality in data mining and time series prediction," in *IWANN'05 Proceedings of the 8th international conference on Artificial Neural Networks: computational Intelligence and Bioinspired Systems*, 2005.

[20] A. Hamza, H. H. Gharakheili, T. Benson, and V. Sivaraman, "Detecting Volumetric Attacks on IoT Devices via SDN-Based Monitoring of MUD Activity," *ACM SOSR*, 2019.

[21] Angela Sasse, "Scaring and Bullying People into Security Won't Work," *IEEE Security & Privacy*, 2015.

[22] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečný, S. Mazzocchi, H. B. McMahan, T. V. Overveldt, D. Petrou, D. Ramage, and J. Roselander, "Towards Federated Learning at Scale: System Design," 2019.