IDSA-IoT: An Intrusion Detection System Architecture for IoT Networks

Guilherme Weigert Cassales,
Hermes Senger
Universidade Federal de São Carlos, Brazil
Email: gwcassales@gmail.com, hermes@ufscar.br

Elaine Ribeiro de Faria
Universidade Federal
de Uberlândia, Brazil
E-mail: elaine@ufu.br

Albert Bifet
University of Waikato, New Zealand
LTCI, Telecom ParisTech, France
Email: abifet@waikato.ac.nz

Abstract—**The Internet of Things (IoT) allows large amounts and variety of devices to connect, interact and exchange data. The IoT network creates numerous opportunities for novel attacks that can compromise information and systems integrity. Intrusion detection systems have been studied over two decades, mostly employing traditional data mining and machine learning techniques that require an offline phase for model training on large amounts of data. This paper presents three data stream novelty detection techniques applied to the intrusion detection problem and proposes IDSA-IoT, a novel implementation architecture, which combines the use of resources at the edge of the network and a public cloud. After an extensive empirical evaluation, results show that it is possible to identify new attack patterns soon after their emergence and to adapt the models in an efficient way.**

I. Introduction

A survey released by Gartner in 2017 estimated that by 2020 there will be about 20 billion devices connected to the Internet, many of them through the IoT [1]. Another survey released by the same company in 2018 estimated that nearly 20% of organizations have experienced at least one IoT-based attack in the last three years. The study shows that most organizations have no control over the origin and nature of software and hardware used by connected devices. To protect against these threats, IoT's worldwide spending on security will increase from $1.5 billion in 2018 to $3.1 billion in 2021 [2], including tools and services to improve asset discovery and management, software security evaluation, hardware testing, and penetration testing.

The so-called Internet of Things (IoT) brings together a wide variety of devices, including mobile, wearable, consumer electronics, automotive and sensors of various types. Such devices use the Internet to connect to other devices, systems, and applications running on the back-end. Once compromised, they can be used to attack other devices and systems, steal information, cause immediate physical damage, or either perform various other malicious actions. Most of them will likely have long lifespan or less frequent software patches. The increase of the mesh of devices of diverse technologies brings with it a considerable increase of the surface of attack. In this scenario, cybersecurity experts and front-line professionals are now tasked with defeating new types of attacks that come with increasing frequency.

Intrusion Detection Systems (IDS) are important tools for protecting corporate networks. From a research point of view, intrusion detection has been a challenge over the years and most of the related work rely on Data Mining (DM) or Machine Learning (ML) techniques to detect attacks from known patterns or to discover new patterns [3], [4]. With traditional data mining methods, the data set is static and can be traversed repeatedly, and the detection of new attack patterns requires a new cycle of training, testing, and dissemination of new models. Unlike traditional methods, stream mining algorithms can be applied to intrusion detection with several advantages, such as: (i) working with limited memory, which allows the implementation in small devices (for example, on the edge from the Web); (ii) processing traffic data with a single read; (iii) producing real-time response; and (iv) detecting novelty and changes in concepts already learned.

Online intrusion detection can be a hard job depending on the number of devices and their physical location. With hundreds or thousands of IoT devices and objects scattered across corporate networks or smart cities, moving the traffic data from the devices where they are collected to be analyzed on a traditional cloud or datacenter can be costly or prohibitive due to high latency. Also, moving torrents of traffic data from a large number of devices to be scanned in a centralized infrastructure is not scalable. The current cloud computing paradigm will hardly be able to meet the requirements of low latency and scalability to support intrusion detection [5]. To face this challenge, it is possible to approximate the intrusion detection function processing to small devices and objects, taking advantage of the resources available on small devices that populate the edge of the network.

In the present work, we propose a distributed intrusion detection system for IoT scenarios with hundreds or thousands of objects and devices. The main contributions are two. First, we propose an intrusion detection architecture that leverages cloud edge capabilities, with the goal of reducing latency and increasing scalability. In addition, we employ and evaluate three Novelty Detection (ND) methods to learn emerging patterns of network traffic. Our proposal combines the use of edge network resources to collect and analyze data streams and public cloud to process offline data for more accurate operations such as model improvements.

This article is organized as follows: Section II presents the

TABLE I
SUMMARY OF RELATED WORKS

Work	Platforms	Technique	DataSet	Metrics
Kasinathan et al [6]	6LoWPAN	Suricata	Real data with metasploit	Accuracy, packets/second
Sheikhan and Bostani [7]	6LoWPAN	Hybrid - MR OPF	NSL-KDD and simulated attacks	FAR and recall
Raza et al [8]	6LoWPAN	DAG analysis	No information	Recall, energy and memory consumption
Furquim et al [9]	WSN	MLP of Weka	Real data	MAE, RMSE, R^2, R, accuracy, recall, precision, specificity
Midi et al [10]	WSN	Independent modules, each with one technique	Trace replay and attack injection	Recall, accuracy, memory and CPU consumption
Lloret et al [11]	Smart City	Clustering, MLP and statistical models	Real data from meters	Water and energy consumption
Diffalah et al [12]	Smart City	LiSA, smoothing function	Real data	Outliers, comparison between simulation and collected data
Faisal et al [13]	*Smart Grid*	7 MOA classifiers	KDD99	Accuracy, Kappa, memory consumption, time, FAR and FNR

related works. Section III reviews methods for detecting new features. Section IV presents the architecture proposal of the present work. Section V presents the architecture validation experiments, encompassing accuracy and processing performance evaluations. Section VI summarizes the main findings and presents possible future work.

II. Related work

6LoWPAN is a standard defined by the IETF in RFC 6282, to transmit data with IpV6 and other protocols on low power wireless devices using IEEE 802.15.4 in the lower layers. However, this technology still lacks protection and security mechanisms. For instance, in [6], signature detection is used to detect DoS and UDP flood attacks. The architecture uses a probe to promiscuously listen the whole traffic of a 6LoWPAN network and sends the data to analysis on a non-constrained host. The work in [8] proposed an hybrid IDS which focuses on specific routing attacks, such as sink-hole and selective-forwarding. Higher complexity tasks which demand more computational resources are executed on the border router, while simpler tasks execute on constrained nodes. Results were expressed by metrics as recall and memory and energy consumption. The work in [7] proposed the use of anomaly detection to identify internal routing attacks, and signature detection to identify external attacks. Anomaly detection was tested with simulated attacks, while signature detection used a subset of NSL-KDD. They used the recall and FAR as metrics.

A three-layer architecture (composed of WSN, Fog and Cloud) with focus on fault tolerance in disaster scenarios is proposed in [9]. Fog computing is used to execute ML functions and data aggregation. Experiments used real data collected from sensors. The metrics used included precision, recall and accuracy. The work in [10] proposed an hybrid IDS which collects information about the environment and activates specific modules to mitigate each kind of attack. Experiments were made in a real environment and metrics used were recall, precision and resource consumption (CPU and RAM). Smart cities scenarios also employ IoT to measuring and monitoring tasks. In [11], the authors propose an architecture that uses three stream mining methods based on ML to characterize water and energy consumption behavior, predict consumption, and detect incidents. The metrics used to express results include water and energy consumption. The work in [12] also aimed to identify anomalies in a water distribution network and proposes a three layer architecture (sensors, base stations and datacenter). The second layer performs time-sensitive tasks, thus reducing latency, while third layer provides storage and aggregates the results of the second layer with historical data to generate more accurate information. Water distribution measures were used, comparing the values of the predictions with the actual measurements. Intrusion detection for smart cities, based on data mining techniques running on an unrestricted devices is proposed in [13]. Experiments using KDD99 data are presented and the metrics used were precision, Kappa, memory consumption, time, FAR, and FNR.

Table I summarizes the discussion on the related work. Note that some works use data from KDD99 or derived from this dataset. Collected two decades ago, KDD is no longer representative of current attack patterns and IoT environments. Some works used traces captured from local infrastructure, which provide realistic evaluation, but lack of reproducibility. Some works use data produced by intentional attacks simulated, designed by the same people who designed the detection techniques. This can bring unrealistic advantages to the detection methods. Also, it is worth noting that most articles used metrics like FAR, recall, and accuracy. Although widely adopted in classical scenarios, such metrics are inaccurate for stream processing [14].

III. Novelty detection

Novelty Detection (ND) is the recognition that an example differs in some way from the previous examples [15]. It allows the recognition of new patterns, which may either trigger the emergence of a new concept, the evolution of a known concept, or the presence of noise [14]. ND offers a valuable technique in many ways when dealing with intrusion detection. For instance, it may be preferable to discover early when some examples are part of an emerging pattern than misclassifying them in a known class. In traditional ML approaches, a new offline step is necessary to retrain the model on new data, to include knowledge about emerging attack patterns in the decision model. This step delays the application of new knowledge and consumes computing power. In addition, supervised learning methods are costly as they require a specialist to label a significant set of examples for model training. On the other hand, ND aims to identify novel and relevant behaviors in the network, and quickly adapt the decision model to recognize them as they emerge.

ND techniques for data streams are usually composed of two phases. The former is the static, offline phase, in which a dataset is labeled and used to learn the known concepts [16]. Three aspects can vary in the offline phase: (*i*) the number of classes associated with known concepts: only normal and attack classes, or discriminating multiple classes of attack; (*ii*) the number of classifiers: single or multiple classifiers, and (*iii*) the learning task, which specifies which algorithms will be used for the construction of the decision model, such as decision trees and KNN, which require the presence of labels for training or clustering techniques, which allows training and updating with or without labels.

During the second, online phase, new continuously arriving data are analyzed. In this phase, three tasks are performed: the classification of new examples, the detection of novelty patterns, and the adaptation of the decision model [16]. The classification consists in analyzing each new example and verifying whether it can be explained by the current decision model. If it can not, it will be considered as unknown and will be analyzed in the future in the search for new patterns.

Finally, in the third step the decision model must be updated, with or without specialist feedback. If specialist feedback is used, then a human expert should label the data, otherwise the model will only add the information of a new found pattern, without knowing if this pattern corresponds to normal or malicious traffic.

While receiving specialized feedback for all examples is difficult to achieve in real-world situations, this indicates a good ceiling for the performance and accuracy of the technique used. On the other hand, the scenario characterized by the absence of feedback by the specialist represents the worst case of this process. A scenario closer to what happens in real life is when a small amount of feedback is provided, and only a subset of examples is made available to the expert. This can be achieved using active learning or semi-supervised techniques.

An example of a technique that uses total feedback is ECSMiner [17], where a set of decision trees is used to represent known concepts. Using the taxonomy proposed in [16], this technique can be described as a classifier capable of recognizing only one new class in the online phase, which uses the unknown label, has a forgetting mechanism, and assumes total expert feedback after a specific time interval. On the other hand, the MINAS [18] and AnyNovel [19] are novelty detection techniques that use a set of clusters to represent the decision model. The main advantages of these approaches is to allow update in the decision model with or without external feedback. Both approaches use a single classifier and a short-time memory to store the examples not explained by the current decision model, the so called unknown examples. These examples are used to update the decision model. The main difference between these approaches is that stable existing concepts aren't updated in AnyNovel.

IV. Proposed Architecture

An architecture that implements intrusion detection based on stream processing techniques is presented below. Intrusion detection is done by stream processing algorithms that run at the edge of the network, close to the IoT devices and sensors. As shown in Figure 1, local resources such as the Single Board Computer (SBC) or routers that have sufficient processing resources are used. Thus, traffic processing is done at the edge of the network, avoiding moving data to the center of the network (in the public cloud, for example). This allows intrusion detection to be done with low latency and avoids the need to send all traffic to be analyzed in the center of the network, such as in large data centers or in the cloud, improving system scalability. Only data that has not been explained by the model and that may be used for evolution of the models must be sent to the public cloud. In the cloud, data from various network regions can be stored until they are analyzed for training of new models or adjustments to existing models.

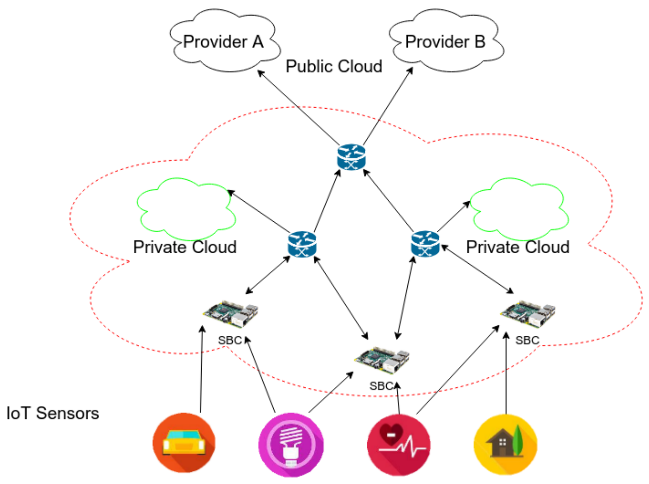

Fig. 1. Physical architecture.

Fig. 2 shows the steps and where they are executed. Fig. 3 presents a more abstract view of the architecture, highlighting the offline and online processing steps.

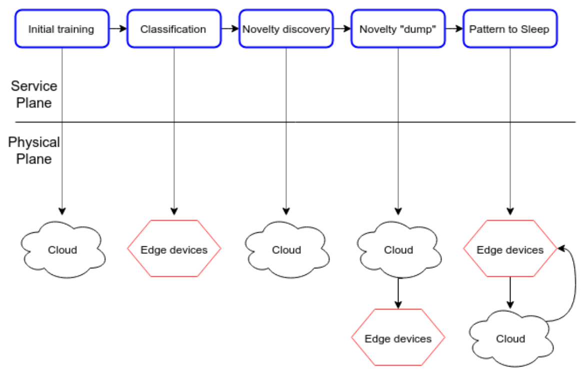

Fig. 2. Service distribution.

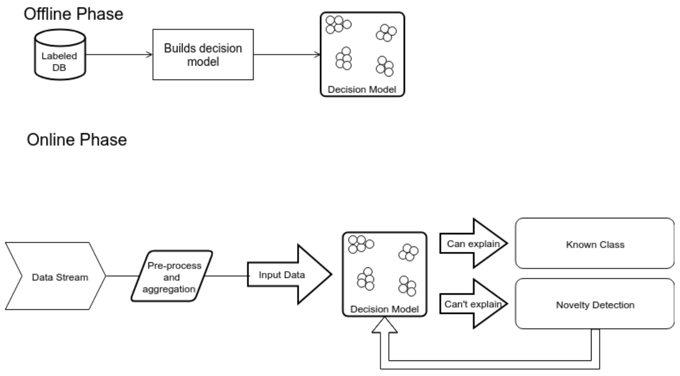

Fig. 3. Logical architecture.

It all starts with an offline step, where decision models are built from labeled data sets. Because this step may require large computational capacity, it runs in the public cloud.

Then, models are disseminated to devices at the edge of the network, where data streams are processed. The first step in the online phase includes preprocessing of data streams, aggregation and classification of data, corresponding to the classification service in Figure 2. These processes execute distributed across multiple devices in the edge of the network. If the example can not be explained by the model, it will be forwarded to the novelty discovery process.

Once the cloud receives a sufficient number of unexplained examples from multiple devices in multiple network locations, the novelty discovery service executes. If valid groups (patterns) are found, the new models must be disseminated to the edge devices that execute ND. Alternatively, a specialist may analyze the novelty patterns discovered before they are spread to the rest of the system. This step corresponds to the knowledge sharing service.

Because network regions may behave differently, some patterns may become outdated in some specific regions. In this case, local models can move these patterns to the sleep memory. Whenever a pattern is sent to the sleep buffer, the edge device must notify the cloud. If a significant amount (defined by a threshold) of edge devices move the same pattern to the sleep buffer, the cloud sends a message ordering all edge devices to move that pattern to the sleep buffer, maintaining a more consistent pattern. This step is represented by the Obsolete Mechanism service.

V. Experiments

For the experiments, we used the Kyoto 2006+ dataset which contains data collected from 2006 to December 2015 [20] [1] . Data were continuously collected from 348 Honeypots of 8 different types including systems with Windows (several releases), Linux / Unix (Solaris 8, MacOS X), network printer, home appliances (eg, TV set, HDD Recorder), to name a few. The dataset has 24 attributes plus four label attributes, three of which were generated by commercial IDS with malware and shell script detection capabilities. The fourth label attribute discriminates against normal, known attacks, and unknown attacks. Most of the attributes are numeric fields and required normalization. However, it is hard to know in advance the highest values that will appear in the stream for attributes like duration, bytes received and bytes sent. For such attributes we fixed maximum values in advance. Nominal attributes were removed.

We selected examples from one month, December, 2015. Only the examples of known attack types and known IDS alert code with a minimum of 10,000 occurrences (for significance) were considered. The offline training was performed with 72,000 examples (i.e., 10% of the dataset) using the holdout technique.

A. Assessing the Quality of Detection

The first set of experiments evaluates the quality of ND. As defined in [17], Fnew measures the percentage of normal class examples incorrectly identified as belonging to a novelty or extension, while Mnew is defined as the percentage of novelty examples erroneously classified as normal class. The error is defined by the quotient between the sum $FN + FP$ and the number of examples classified. Another metric used was the amount of expert-labeled examples that each technique requested. Three ND techniques which implement total or partial feedback were evaluated.

We have performed experiments aiming to evaluate the technique ECSMiner with varying amount of labels delivered while maintaining the delay. Fig. 4 shows the standard behaviour of the technique, in which, every label will be available to the model with a delay of 50,000 examples. The Figures 5 and 6 show a modified version, which tries to approximate the conditions of a real scenario where the amount of labels delivered are, respectively, 20% and 1%.

Perhaps the most visible difference among the figures is the amount of vertical lines. The less labels the model gets, the more lines appear. On Fig. 4 the lines are between timestamps 0 and 50,000, which means that although the model can't explain the examples, it can't update itself because it doesn't have labels yet, thus an alert is raised. When the labels begin to arrive, the model tries to incorporate this new class, causing an increase in Fnew. Since this is a set of binary data, the type of attack is indifferent, and therefore the DoS and Scan attacks have the same class although they have different characteristics. This feature of the dataset caused the model to incorrectly classify examples of a known class as novelties. Notice how the lines go further than timestamp 50,000 on Fig. 5 and 6. This happens because ECSMiner uses labeled examples to train a new tree for the ensemble, therefore if the model is receiving less labels this process will take longer.

Based on these experiments, it is possible to say that tree ensemble is a good technique to classify security datasets. Given that the Fnew measure was smaller than 3% on all experiments and Error was also below 5%. The results were stable even with fewer labels available, however, the amount of feedback needed was still high (7,200 labeled examples on Fig.

[1] Available at: http://www.takakura.com/Kyoto_data/new_data201704/

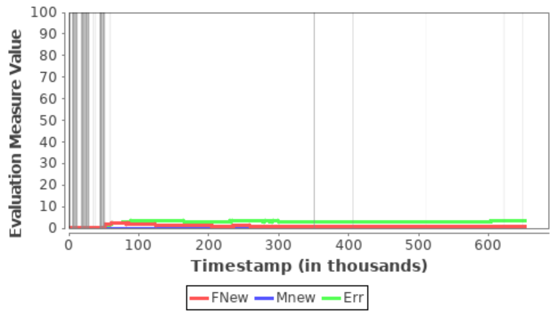

Fig. 4. Binary classification with a label delivery delay of 50,000 examples and total feedback (ECSMiner).

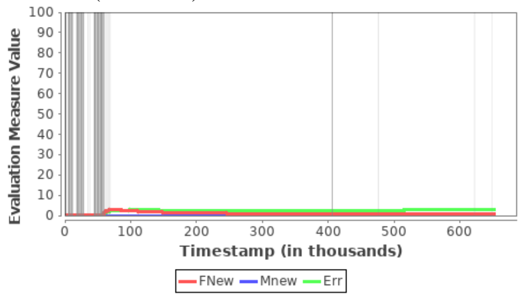

Fig. 5. Binary classification with a label delivery delay of 50,000 examples and partial feedback of 20% (ECSMiner).

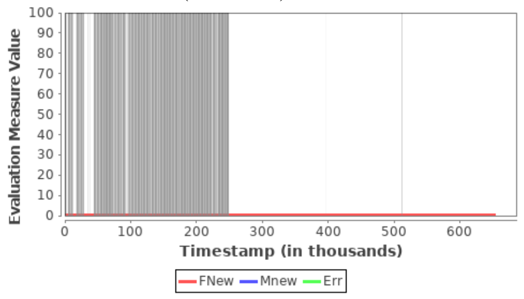

Fig. 6. Binary classification with label delivery delay of 50,000 examples and partial feedback of 1% (ECSMiner).

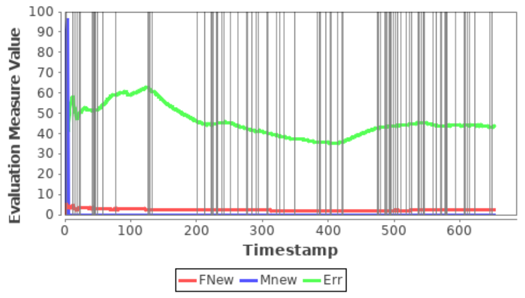

Fig. 7. Binary classification, clustering algorithm with partial feedback of 0.023% (MINAS).

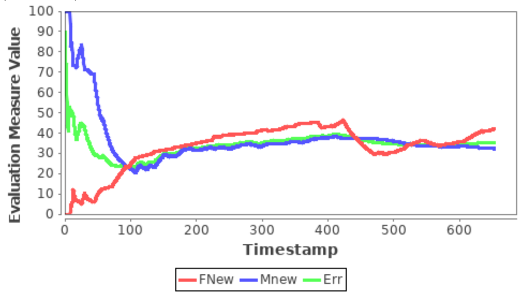

Fig. 8. Binary classification, clustering algorithm with partial feedback of 0,041% (AnyNovel).

6 and 144,000 on Fig. 5). Another downside of this method is that the processing time was very slow if compared to other techniques (more information on Subsection V-B). Therefore, we sought other techniques that could be processed faster in a constrained device and required less specialist feedback.

We evaluated two cluster-based techniques with partial feedback, MINAS and AnyNovel. The selected results are shown in Figures 7 and 8.

Fig. 7 shows a Fnew of approximately 5%, we have performed experiments with other parameters and MINAS maintained this rate of Fnew on most of them, suggesting that this method is robust regarding the ND (compared to AnyNovel which presented a wide variation in measures), but has to be improved regarding the error rate. AnyNovel result, on the other hand, shows a behaviour where the Fnew is, most of the time, growing. This indicates the technique is having difficulties to correctly identify the novelties. Another scenario is possible with AnyNovel where Fnew is less than 2% for the whole execution, however, in this case, Mnew is higher than 90%, indicating a tradeoff between a accuraccy in detection and volume of detection.

Also, there is another tradeoff involving the amount of labeled examples needed and the type of model used. As expected, a low amount of labeled examples generates a larger error and Fnew, while high amounts of labeled examples generate smaller error and Fnew. The ensemble of trees achieved better results, however it needs more labels than both clustering approaches. Finally, the clustering approaches can react more quickly to flow variations, regardless of the labels provided.

The high error observed in Fig. 8 suggests that AnyNovel is not suitable for intrusion detection, however, they provide useful insights. AnyNovel has 10 parameters, of which we chose to fixate the four labels, leaving only six parameters. From these six parameters, we chose to use two values in five of them and three values in the last one. The values used were: $SegmentSize$ 200, 500 and 2000. $Slacks$ 0.1 and 0.9. $Movement$ 0.1 and 1.0. $StableSize$ 20 and 50. $AwayThreshold$ and $NovelSlack$ 0.1 and 0.6. These configurations totaled 96 experiments, from which we extracted the most interesting results to present below.

Table II shows a summary on the Fnew, Mnew, Error and execution time (executed on notebook described in Subsection V-B). An x in place of the parameters means this parameter had no impact in result. There is an inverse relation between Fnew and both Mnew and Error. On experiment 200-0.1-0.1-50-0.6 Fnew is below 2%, the smallest achieved, however Mnew is higher than 95% and error is over 60%. On the other hand, experiment 500-0.9-x-x-x (presented on Fig. 8) shows Fnew around 42% while Mnew and Error stabilize at 32 and

35%. These experiments had opposite parameters, therefore it is easier to analyze each parameter individually for better understanding.

The most influential parameter is the size of a $Slack$ boundary around the concept clusters. Higher values in this parameter allow the model to classify examples in an existing concept even if they are further away, therefore providing robustness to data dispersion. Bigger $Slacks$ value provided superior results. This suggests that Kyoto dataset has a high dispersion, and thus, a model capable of accommodating higher distances produces better results. The parameter *Away Threshold* has also provided better results with a higher value. This parameter denotes how distant from the model a cluster has to be to be recognized as a novelty, therefore, smaller values will output a higher number of novelties. Better results with higher value concurs with $Slacks$ findings that the dataset has high dispersion.

Among the other parameters, only the *Segment Size*, which configures the size of each window, produced noticeable difference in performance. In general, a window of 200 examples had two downsides, (i) the processing time was higher than the on bigger windows; (ii) the quality of detection was smaller in all cases where only the window changed. This finding indicates the technique is very volatile, making the differentiation of behaviors more difficult with less data. This volatility is reinforced by experiment 200-0.1-0.1-50-0.6 where the algorithm can detect the more obvious novelties and ends up missing most of the novelties which are more similar to the normal behaviour. When using more data the algorithm is capable of identifying more patterns and thus, reduces Mnew and Error at the cost of producing more false positives (Fnew). Based on these findings, it is safe to hypothesize that attribute selection techniques might improve the results of this technique on this dataset.

TABLE II
SUMMARY OF ANYNOVEL RESULTS

SegmentSize-Slacks -Movement-StableSize -AwayThreshold	Fnew	Mnew	Error	Time(s)
2000-0.1-x-x-x	17.75%	65.15%	50.18%	129.670
2000-0.9-x-x-x	25.32%	52.78%	44.10%	143.612
500-0.1-x-20-x	18.35%	66.40%	51.10%	152.639
500-0.1-x-50-x	20.83%	60.01%	47.53%	163.291
500-0.9-x-x-x	41.95%	32.23%	35.30%	337.103
200-0.1-0.1-50-0.6	1.93%	95.58%	64.19%	532.388
200-0.1-0.1-50-0.1	34.42%	49.42%	44.69%	422.720
200-0.1-1.0-50-0.1	11.66%	76.69%	56.09%	228.172
200-0.9-1.0-20-x	23.05%	59.33%	47.86%	735.232
200-0.9-0.1-20-x	23.60%	59.25%	47.97%	702.040

Experiments with smaller values of the *Slacks* parameter have shorter execution times, showing a tradeoff between speed and quality of results. Another parameter of large influence on the execution time is *Segment Size*, the smaller the segment the slower the execution. Another important result to contrast with the next subsection is that, on standalone experiments, AnyNovel memory consumption reached almost 1Gb.

B. Performance in Constrained Devices

This experiment aims at evaluating the feasibility of evaluated methods running on devices with constrained resources. We implemented an isolated network with a Switch 3Com Baseline 2824 with 24 10/100/1000 ports, a Raspberry Pi and a notebook with a Core i5-7200U 2.5GHz processor and DDR4 2133MHz 8GB memory.

A producer process running on the notebook replayed the contents of the dataset in fthe form of data streams with controlled parameterized rates. The data was posted on a Kafka server running on the same notebook. A Raspberry Pi 3 card consumed Kafka data and performed classification and ND.The producer has three parameters (R, S, I), where R (Base rate) is the starting rate of examples published in a kafka topic per second. Every I seconds, the rate will be increased by S units, until 80MB of data is generated. In Table III we present, for each technique, the Base Rate that produced the best execution time as well as the amount of memory it consumed. Every experiment in this table used $S = 10$ and $I = 10$. The base rate was increased by 100 until the execution time stagnated – once the producer's rate surpasses the consumer's speed examples will start to accumulate on the topic, however this has no impact in consumer performance. The memory collection technique used ps auxiliary libraries and took the required precautions to guarantee the only JVM process running on Pi was the classifier of choice. Each technique was configured with the parameters that resulted in the fastest execution time.

TABLE III
SUMMARY OF EXPERIMENTS

Technique	Max Rate	Time (s)	Memory (Mb)
MINAS	2300	60.579	65.1
AnyNovel	5100	121.029	268.6
ECSMiner	500	764.426	146.1

MINAS technique has the smallest memory consumption and execution time, thanks to using the simplest model and a very straight forward classification. Although AnyNovel's rate is more than twice the rate of MINAS, its execution time is worse. That's because AnyNovel processes data in windows, creating an idle period while accumulating data for each window and increasing memory consumption.

These results also indicate a trade-off between quality and speed of classification. On the previous experiments ECSMiner showed the best metrics regarding accuracy, however, its execution time is more than 10 times slower than MINAS, while the rate used is less than 5 times bigger.

C. Summary of Experiments

In summary, these experiments provided insights about the techniques and the dataset.

In a scenario where processing power and specialist feedback are abundant, ECSMiner is the best option. However, neither of these requisites is easily fulfilled in IoT scenarios. The second smallest memory consumption , indicates a good alternative if the classification performance could be improved.

MINAS technique presented high error rates. However, the small and constant Fnew coupled with the smallest execution time and memory consumption, indicates MINAS is suitable for IoT scenarios. Also, the error rate is possible to be improved by adjusting the parameters.

Finally, AnyNovel has good execution time, and the memory consumption on the experiments with Kafka improved compared to standalone experiments. However the achieved results varied too widely, which might be improved with better parametrization or feature selection. Yet there was no configuration on our tests that resulted in a good metric in one category while being acceptable on the others.

VI. Conclusion

In this work, an intrusion detection system architecture for IoT environments was presented. The architecture makes combined use of a large computational capacity infrastructure, such as the public cloud, and distributed resources at the edge of the network. Classification is performed at the edge near devices and sensors to reduce processing latency, while the cloud is used for the enhancement and dissemination of models so that the elements at the edge are always up to date. The physical schema allows the system to be used for different types of IoT networks, but is able to maintain control over the distribution of tasks. The proposed logical scheme is generic and can be adapted to other ND techniques. Experiments demonstrate the feasibility of the ND technique applied in the context of intrusion detection. Some trade-offs have been identified for the amount of feedback required with respect to the type of model used and the quality of the results. It is known that in a context close to reality, the amount of labels available will be small, so it is interesting that the methods use as little as possible to generate good results. Experiments have shown that a device with reduced resources is capable of performing the classification and detection of novelties with a flow rate of 450Kb per second.

As future work we intend to test this architecture with a wider range of techniques as well as improve some of the techniques used and/or implement a novel technique. Also, as a possible future work we highlight the possibility of a comparative analysis of performance using a larger amount of data.

Acknowledgment

The authors thank CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Código de Financiamento 001). Hermes Senger also thanks CNPQ (Contract 305032/2015-1) and FAPESP (Contract 2018/00452-2, and Contract 2015/24461-2) for their support.

References

[1] "Gartner says 8.4 billion connected "things" will be in use in 2017, up 31 percent from 2016," 2017. [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/2017-02-07-gartner-says-8-billion-connected-things-will-be-in-use-in-2017-up-31-percent-from-2016

[2] "Gartner says worldwide iot security spending will reach $1.5 billion in 2018," Mar 2018. [Online]. Available: https://www.gartner.com/newsroom/id/3869181

[3] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 2, pp. 1153–1176, 2016.

[4] R. Mitchell and I.-R. Chen, "A survey of intrusion detection techniques for cyber-physical systems," *ACM Computing Surveys (CSUR)*, vol. 46, no. 4, p. 55, 2014.

[5] A. V. Dastjerdi and R. Buyya, "Fog computing: Helping the internet of things realize its potential," *Computer*, vol. 49, no. 8, pp. 112–116, 2016.

[6] P. Kasinathan, C. Pastrone, M. A. Spirito, and M. Vinkovits, "Denial-of-service detection in 6lowpan based internet of things," in *IEEE 9th Intl. Conf. on Wireless and Mobile Computing, Networking and Communications (WiMob)*, Oct 2013, pp. 600–607.

[7] M. Sheikhan and H. Bostani, "A hybrid intrusion detection architecture for internet of things," in *8th Intl. Symp. on Telecommunications (IST)*, Sept 2016, pp. 601–606.

[8] S. Raza, L. Wallgren, and T. Voigt, "Svelte: Real-time intrusion detection in the internet of things," *Ad Hoc Networks*, vol. 11, no. 8, pp. 2661 – 2674, 2013.

[9] G. Furquim, G. P. R. Filho, R. Jalali, G. Pessin, R. W. Pazzi, and J. Ueyama, "How to improve fault tolerance in disaster predictions: A case study about flash floods using iot, ml and real data," *Sensors*, vol. 18, no. 3, 2018.

[10] D. Midi, A. Rullo, A. Mudgerikar, and E. Bertino, "Kalis - a system for knowledge-driven adaptable intrusion detection for the internet of things," in *IEEE Intl. Conf. on Distributed Computing Systems (ICDCS)*, June 2017, pp. 656–666.

[11] J. Lloret, J. Tomas, A. Canovas, and L. Parra, "An integrated iot architecture for smart metering," *IEEE Communications Magazine*, vol. 54, no. 12, pp. 50–57, 2016.

[12] D. E. Difallah, P. Cudré-Mauroux, and S. A. McKenna, "Scalable anomaly detection for smart city infrastructure networks," *IEEE Internet Computing*, vol. 17, no. 6, pp. 39–47, Nov 2013.

[13] M. A. Faisal, Z. Aung, J. R. Williams, and A. Sanchez, "Data-stream-based intrusion detection system for advanced metering infrastructure in smart grid: A feasibility study," *IEEE Systems Journal*, vol. 9, no. 1, pp. 31–44, March 2015.

[14] J. Gama, *Knowledge Discovery from Data Streams*. CRC Press Chapman Hall, 2010.

[15] P. Perner, "Concepts for novelty detection and handling based on a case-based reasoning process scheme," in *Advances in Data Mining. Theoretical Aspects and Applications*. Springer, 2007, pp. 21–33.

[16] E. R. Faria, I. J. C. R. Gonçalves, A. C. P. L. F. de Carvalho, and J. Gama, "Novelty detection in data streams," *Artificial Intelligence Review*, vol. 45, no. 2, pp. 235–269, Feb 2016.

[17] M. Masud, J. Gao, L. Khan, J. Han, and B. M. Thuraisingham, "Classification and novel class detection in concept-drifting data streams under time constraints," *IEEE Trans. on Knowledge and Data Engineering*, vol. 23, no. 6, pp. 859–874, June 2011.

[18] E. R. de Faria, A. C. Ponce de Leon Ferreira Carvalho, and J. Gama, "Minas: multiclass learning algorithm for novelty detection in data streams," *Data Mining and Knowledge Discovery*, vol. 30, no. 3, pp. 640–680, May 2016.

[19] Z. S. Abdallah, M. M. Gaber, B. Srinivasan, and S. Krishnaswamy, "Anynovel: detection of novel concepts in evolving data streams," *Evolving Systems*, vol. 7, no. 2, pp. 73–93, 2016.

[20] J. Song, H. Takakura, Y. Okabe, M. Eto, D. Inoue, and K. Nakao, "Statistical analysis of honeypot data and building of Kyoto 2006+ dataset for nids evaluation," in *Proc.of Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*. New York, NY, USA: ACM, 2011, pp. 29–36.