# IoT Data Stream Novelty Detection: Design, Implementation and Evaluation [DRAFT]

Luís Puhl, Guilherme Weigert Cassales, Hermes Senger, Helio Crestana Guardia

Universidade Federal de São Carlos, Brasil
https://orcid.org/{0000-0003-4029-2047, 0000-0003-1273-9809, 0000-0001-5010-747X}

***Abstract*—The ongoing implementation of the Internet of Things (IoT) is sharply increasing the small devices count and variety on edge networks and, following this increase the attack opportunities for hostile agents also increases, requiring more from the network administrator role and the need for tools to detect and react to those threats. One such tool are the Network Intrusion Detection Systems (NIDS) where the network traffic is captured and analysed raising alarms when a known attack pattern or new pattern is detected. For a network security tool to operate in the context of edge and IoT it has to comply with processing time, storage space and energy requirements alongside traditional requirements for stream and network analysis like accuracy and scalability. This work addresses the construction details and evaluation of an prototype distributed IDS using MINAS Novelty Detection algorithm following up the previously defined IDSA-IoT architecture. We discuss the algorithm steps, how it can be deployed in a distributed environment, the impacts on the accuracy of MINAS and evaluate the performance and scalability using a cluster of constrained devices commonly found in IoT scenarios. We found an increase of *A 0.0* processed network flow descriptors per core added to the cluster. Also *B 0.0%* and *C 0.0%* change in *F1Score* in the tested datasets when stream was unlimited in speed and limited to *0.0 z MB/s* respectively.**

***Index Terms*—novelty detection, intrusion detection, data streams, distributed system, edge computing, internet of things**

## I. Introduction

Atualizar o iscurso e tirar um pouco o foco de IDS para detecção de novidades na rede

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[**?**]. 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 [**?**], 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 [**?**], [**?**]. 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 [**?**]. 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.

refazer este paragrafo: O presente trabalho avança a pesquisa sobre um trabalho anterior [ISCC-2019], apresentando as seguintes contribuições: (i) a arquitetura proposta em [ISCC] foi instanciada e validada de forma experimental, (ii) avaliamos o impacto da distribuiçõão do processamento sobre a qualidade da detecção de novidades quanto da eficiência do processamento (iii) discutimos estratégias de distribuição de fluxos para classificação, incluindo a detecção de novidades foram discutidas

CNPq

TABLE I: Summary of related works

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

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

Expected results: A system that embraces and explores the inherent distribution of fog computing in a IoT scenario opposing regular systems where data streams are collected and centralized before processing; Impact assessment of the impact of distributed, regional flow characteristics, local vs global vs distributed forgetting mechanism and other polices.

IDS characteristics and description of physical scenario.

MINAS characteristics.

Distribution and IDSA-IoT architecture.

This paper is structured as follows: Section II presents previous works that addresses related problems and how they influenced our solution. Section IV-B address our proposal, the work done, issues found during implementation and discusses parameters and configurations options and how we arrived at our choices. Section V shows experiments layouts and results, we compare serial and distributed implementation's metrics for validation, we also evaluate communication delay effects on classification metrics and conclude with the speedup per core and overall maximum stream speed. Section VI summarizes the research results and presents our final conclusions and future works.

## II. Related Work Nao mexer por enquanto

Recent works explored those areas, to name a few: BigFlow [1] employing Apache Kafka and Apache Flink for distributed stream processing evaluating with package stream dataset, CATRACA [2], [3] uses Apache Kafka and Apache Spark for stream processing and

[4]

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 [**?**], 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 [**?**] 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 [**?**] 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 [**?**]. 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 [**?**] 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 [**?**], 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 [**?**] 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 [**?**]. 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 [**?**].

## III. MINAS GUILHERME PODE FAZER

Apresentar o MINAS de forma resumida (1 pagina no max): como funciona, etc. Veja se a Quali do Guilherme tem algum techo que possa inspirar

Citar que o trabalho anterior já validou o uso do MINAs para detecção de novidade porem com uma implementação sequencial.

Figura do MINAS (offline + online) ...

## IV. PROPOSAL

Este apresenta uma proposta de implementação distribuida do MINAS que segue as diretrizes da arquitetura IDSA-IoT, atendendo aos seguintes requisitos: (i) a etapa de classificação de vários fluxos deve ocorrer em paralelo, sendo processada em diversos locais físicos da arquitetura (ii) a etapa de detecção de novidades (evolução do modelo) deve ocorrer em paralelo, sendo processada em diversos locais físicos da arquitetura (iii) as duas etapas anteriores, por sua vez também deverão ser executadas paralelamente, podendo ocorrer em partes distintas do sistema (iv) possa ser implementada em dispositivos com recursos limitados

Relembrar ids-iot, fase offline e online

Thus we propose a NIDS using MINAS [5] (a Novelty Detection algorithm) to effectively identify previous and new intrusion threats, implemented over a architecture [6] using parallel and distributed techniques leveraging edge and cloud for efficient computing.

[start] intro/related?

NIDS monitor the packet network traffic, aggregate into flow descriptors and analyze to identify any intrusion or misbehavior. However, this problem requires both fast and accurate response: the former is needed to have a proper reaction before harm can be cast to the network and to cope with the traffic without imposing loss or delay in the NIDS or observed network; the latter is required as to not misidentify harmless with harmful and vice-versa. To achieve those goals we leverage fog computing.

In common IoT scenarios, data is captured by small devices and sent to the cloud for any compute or storage tasks, however this is not feasible in our NIDS scenario, even though we also capture data produced in the edge, sending this data to the cloud would in the worst case double the internet communication requirements of the overall system. Fog computing infrastructure aims to offload computing resources from cloud providers by placing edge devices closer to end-users and/or data sources. But two MINAS steps limit this fog offload, the processing intensive novelty detection and, long term model storage and distribution of the internal model. Those steps surpass the capabilities of common fog hardware and therefore need to be at least shared to a cloud where such requirements are easy and cheap to fulfill.

[end] intro/related?

In our proposed NIDS, fog and cloud computing resources are employed as to minimize the time elapsed between a flow descriptor ingestion and intrusion alarm, allocating the classification step of MINAS in a MPI cluster running multiple classifier instances. After the initial classification, the resulting label can be used immediately, but if the sample is labeled as *unknown*, this sample must be stored and the novelty detection step will be triggered, and those steps require more resources and thus are divided in fog and cloud.

1a visão geral de iot nessa fig quais redes envolvidas processamento extra na borda

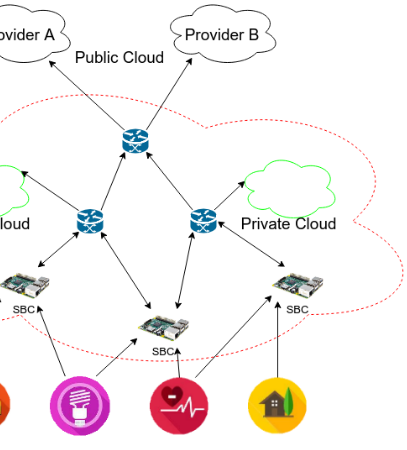
1b enumerar módulos e funções descrever etapas do minas e onde se encaixa descrever onde módulos são multiplexados descrever casos de muiltiplos fluxos (redes locais)

The overall organization of components, connections and interactions with external actors is shown in Figure 1a, from bottom left to top right: sensor network; fog containing gateway router and novelty detection cluster; cloud storage for model, alarms and statistics and; human supervisor addressing alarms and statistics.

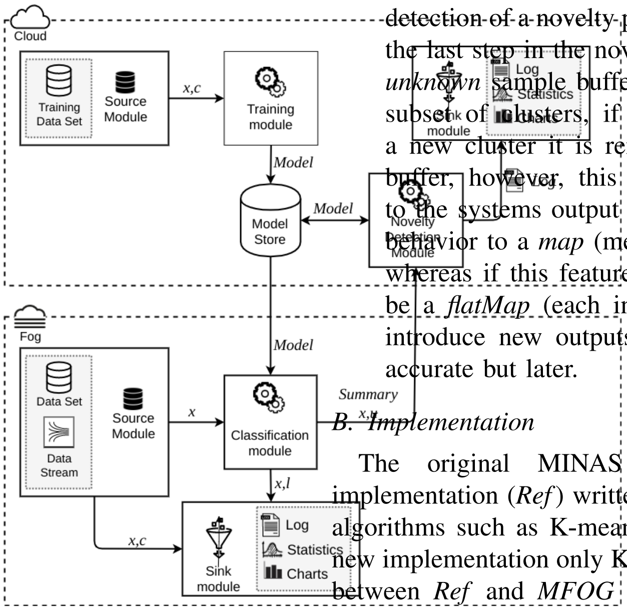
In Figure 1b we depict each logical component associated with each MINAS step and its communications, we also depict extra modules for sampling and measurements. Each communication in Figure 1b shows the direction of the data flow and identifies the data contained: $Model$ is MINAS internal Model containing a set of cluster data structures, $x, c$ identifies a sample with the real class, $x$ is the sample without the real class, $x, l$ identifies a sample with the assigned label, $x, u$ is sample with the *unknown* label, $summary$ is a statistical summary of model usage.

### A. Polices

The distribution of steps and tasks in various modules opens data distribution and its impacts to discussion. The



ecture and deployment scenario overview.



(b) *MFOG* components and communications overview.

Fig. 1: Architecture overview.

decisions following these discussions can be organized in several policies, some of them are:

- Regarding the allocation of the Novelty Detection Module:
  - It can be located at each fog node meaning novelties will be only detected if sufficient patterns occurs in the local observed network, it also spends the local node processing power and a model sharing mechanism must be added;
  - It can be located in the cloud and thus detect patters even when their footprint is small in each local network, also a model sharing mechanism is not needed as the model has a single instance stored in the cloud, the penalty of this choice is increased internet usage as any sample with *unknown* label must be sent from edge to cloud, implying some delay between the appearance of a novel pattern, its detection and its propagation to fog classifiers;
  - It can be located in both, meaning that a local *unknown* buffer is maintained, novelty detection is performed on that buffer, and once a sample is to be discarded as noise or outlier it must be sent to the cloud where the process repeats but with global data. This choice also need the model sharing mechanism and is clearly the more complex.
- Regarding the model cleanup (forget mechanism): Even when a global novelty detection is used, local models can be optimized for faster classification using the its local model statistics, sorting last or removing clusters that are not in frequent use;
- Lastly, a feature not explicitly shown in the original MINAS is the reclassification of *unknowns* after the detection of a novelty pattern with the newfound label: As the last step in the novelty detection step in MINAS, the *unknown* sample buffer is classified using the newfound subset of clusters, if the sample can be explained by a new cluster it is removed from the *unknown* sample buffer, however, this new labeling is not put forward to the systems output restraining the system data-stream behavior to a *map* (meaning each input has one output), whereas if this feature was enabled the behavior would be a *flatMap* (each input can have many outputs) and introduce new outputs, more recent and perhaps more accurate but later.

### *B. Implementation*

The original MINAS algorithm has a companion implementation (*Ref*) written in Java using MOA library base algorithms such as K-means and CluStream, however in the new implementation only K-means is used. Another difference between *Ref* and *MFOG* is cluster radius calculation from the distances of elements forming the cluster and the cluster's center, where the former uses the maximum distance, the latter uses the standard deviation of all distances as described in [5].

The stream format for input and output also of note. Input information needed is the value of the item, this value is a number sequence of length $d$ (referenced as dimension). In addition to the value for evaluation and training purposes the class identifier as single character, optionally an unique item identifier (*uid*) can be provided. For output information and format the decision isn't so clear as we can't predict future system integrations needs like only novelty alarms or every item's original value with assigned label so, we have a compromise and put only enough information for the Evaluation Module (where the full information from the testing file or stream can be accessed) meaning the format can be defined as a tuple containing *uid* and assigned label.

Another implementation decision related to the output stream is whether or not to reprocess, and add to the output stream, examples in the unknown buffer after the novelty detection procedure, meaning one item can be classified once as unknown and again with a label. Our preliminary tests using this technique had increased true positives when compared to not using it. However this changes the stream operator behavior from a *Map* to a *FlatMap* having duplicate entries on the output stream as previously mentioned. Regardless of choice the classification of the unknown buffer after a model update, using the full model or just the added set of clusters, is done to remove the examples "consumed" in the creation of a new cluster in the internals of the clustering algorithm.

For *MFOG* the Message Passing Interface (MPI) library was used. In MPI programming, multiple processes of the same program are created by the runtime and each process instance receives a rank parameter, for *MFOG* this parameters indicate if the process is root, rank $0$, or leaf otherwise. Beyond this division, each process also operates two threads, for the root there is a sampler and detector threads, for the leafs each has a model receiver thread and multiple classifier threads. The overall sequence of interactions is shown in Figure 2.

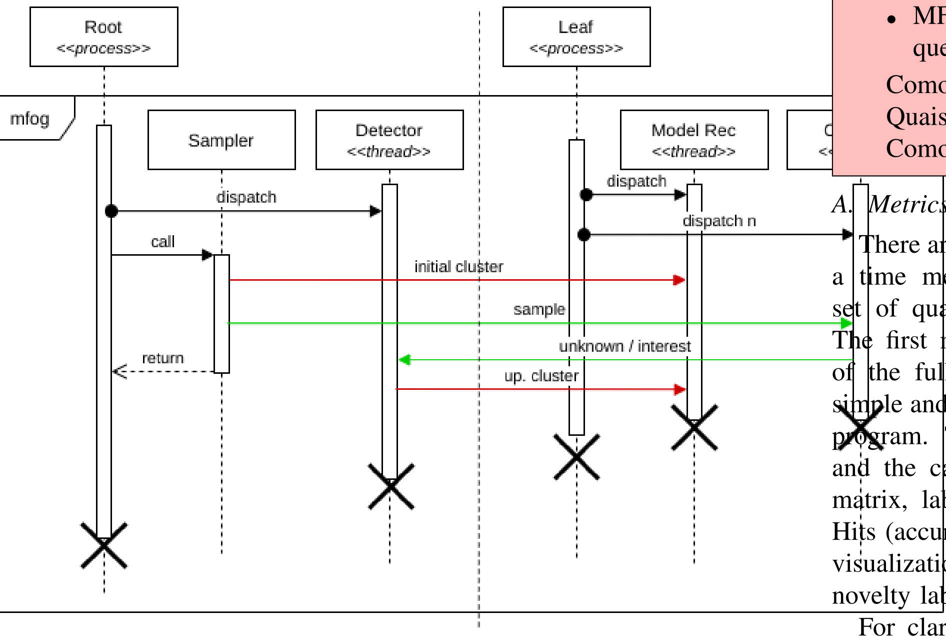


Fig. 2: *MFOG* life line overview.

The Evaluation Module was also build following reference techniques like multi-class confusion matrix with label-class association [5] to extract classification quality metrics.

## V. Experiments and Results

For the experimental setup we dedicated three Raspberry Pi 3 model B single board computers connected via Gigabit Ethernet Switch forming a simple cluster. This cluster stored all source code, binaries (compiled and linked in place) and datasets, being accessed via our laboratory network over Secure Shell (SSH). All experiments were executed in this cluster for isolation of otherwise unforeseen variations.

The dataset used is the December 2015 segment of Kyoto 2006+ Dataset[1] (Traffic Data from Kyoto University's Honeypots) [7]. This segment was filtered (from 7 865 245 instances) to only examples associated to known attack types identified by existing IDS, and attack types with more than 10 000 instances for significance as done by [6]. The remaining instantes then were transformed by normalization, transforming each feature value space (e.g. IP Address, Duration, Service) to the Real interval $[0, 1]$. The result is stored in two sets, training set and test set, using the holdout technique filtering in only normal class resulting in 72 000 instances for training set and 653 457 for test set, containing 206 278 $N$ (normal) class and 447 179 $A$ (attack) class.

O que quer testar com os experimentos.

- Tese: Mostrar que detecção por novidade e classificação continua viável em fog.
- Seria inviável por conta do atraso de distribuição de modelo e,
- limitação pelo hardware pequeno.
- MFOG: Um Agregador Regional, instalado na FOG, que observa a rede local.

Como realizou (cenário, rpi, setup, coleta de métricas).
Quais resultados obteve.
Como interpretar os resultados.

### *A. Metrics and Visualizations*

There are two broad evaluation metrics for each experiment: a time mesure extracted by using *GNU Time 1.9* and, a set of qualitative mesures extracted by a python program. The first metric is straightforward and is the time measure of the full program execution. The latter metric is not as simple and for its extraction required a purposely build python program. This program takes two inputs, the test dataset and the captured output stream, and outputs the confusion matrix, label-class association, final quality summary with: Hits (accuracy), Misses (Err), Unknowns (UnkR); and stream visualization chart with per example instance summary with novelty label markers.

For clarity, it is necessary to detail how to interpret and compare each metric, as for some it is trivial but others are not so straightforward.

In the confusion matrix $M = m_{ij} \in \mathbb{N}^{c \times l}$, computed by our evaluation program, each row denotes one of the datasets original (actual) class and each column denotes the marked (predicted) label present in the captured output stream. Thus, each cell $M_{c,l}$ contains the count of examples from the test dataset of class $c$ found in the output stream with the label $l$ assigned by the under evaluation experiment. For the dataset under use, original classes are *"N"* and *"A"*, and for the labels we have the training class *"N"*, unknown label *"-"* and the novelties $i \in \mathbb{N}$.

Added to the original confusion matrix $C$ are the rows *Assigned* and *Hits*. The former represents which original class $c$ (or if unknown, -) the label $l$ is assigned to, this is computed by using the original class if $c = l$ or by associated novelty label to original class as described in [8] section 4.1. The latter row, *Hits*, shows the true positive count for each label, computed by coping the value of the cell $M_{c,l}$ where the label is the same and the class $c$ is the value in the above *Assigned* row. The *Hits* row is also used to compute the overall accuracy.

For the metric summary table, six metrics from two sources are displayed. Three metrics *Hits Unknowns Misses* represented as ratio of the captured output stream, extracted from the evaluation python program, computed as follows: *Hits* (overall accuracy) is the summation of the homograph row in the extended confusion matrix; *Unknowns* is the count of examples in the captured output stream marked with the unknown label -; *Misses* is the count of all examples in the captured output stream marked with a label distinct from the *Assigned* original class and are not marked as unknown. *Time*, *System* and *Elapsed* represented in seconds, are extracted from *GNU Time*. *Time* is the amount of CPU

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

seconds expended in user-mode (indicates time used doing CPU intensive computing, e.g. math); *System* is the amount of CPU seconds expended in kernel-mode (for our case it indicates time doing input or output); *Elapsed* is the real-world (wall clock) elapsed time (indicates how long another system or person had to wait for the result). To compare the time metric is simple, the lower time taken, the better.

Lastly, the stream visualization chart shows the summary quality metrics (*Hits Unknowns Misses*) computed for each example in the captured output stream. This summary is computed for each example but it uses the *Assigned* row computed previously to evaluate *Hits*, other metrics are derived as described before. Therefore, horizontal axis (x, domain) plots the index of the example and the vertical axis (y, image) shows the metric computed until that example index on the captured output stream. Adding to the summary metrics, novelty label markers are represented as vertical lines indicating *when* in the captured output stream a new label first appeared. Some of the novelty label markers include the label itself ($l \in \mathbb{N}$) for reference as if showing every label would turn this feature unreadable due to overlapping.

### B. Results Discussion

Four main experiments need detailed discussion: (a) reference implementation of Minas (*Ref*) [5]; (b) new implementation in serial mode; (c) new implementation in single-node, multi-task mode and (d) new implementation in multi-node, multi-task mode. Each experiment uses the adequate binary executable, initial model (or training set for the reference implementation) and test set to compute a resulting output stream which is stored for qualitative evaluation. The summary of all four experiments is shown in Table II.

TABLE II: Experiments Collected Metrics Summary

| | Exp. (a) | Exp. (b) | Exp. (c) | Exp. (d) |
|---|---|---|---|---|
| Hits | 0.305618 | 0.298438 | 0.312416 | 0.312478 |
| Misses | 0.676049 | 0.657843 | 0.664061 | 0.663802 |
| Unknowns | 0.018333 | 0.043717 | 0.023521 | 0.023718 |
| Time | 2761.830000 | 80.790000 | 522.100000 | 207.140000 |
| System | 7.150000 | 11.510000 | 47.770000 | 157.610000 |
| Elapsed | 2772.070000 | 93.030000 | 145.040000 | 95.380000 |

The first two experiments (a and b) comparison does serve as validation for our implementation, while the latter three (b, c and d) serves as showcase for the effects of distribution.

## VI. Conclusion Nao mexer por enquanto

## Acknowledgment

The authors thank CNPq (Contract 167345/2018-4). 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] E. Viegas, A. Santin, A. Bessani, and N. Neves, “Bigflow: Real-time and reliable anomaly-based intrusion detection for high-speed networks,” *Future Generation Computer Systems*, vol. 93, pp. 473 – 485, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X18307635

[2] M. E. Andreoni Lopez, “A monitoring and threat detection system using stream processing as a virtual function for big data,” Theses, Sorbonne Université ; Universidade federal do Rio de Janeiro, Jun 2018. [Online]. Available: https://tel.archives-ouvertes.fr/tel-02111017

[3] M. E. Andreoni Lopez, “A Monitoring and Threat Detection System Using Stream Processing as a Virtual Function for Big Data,” Ph.D. dissertation, 2019.

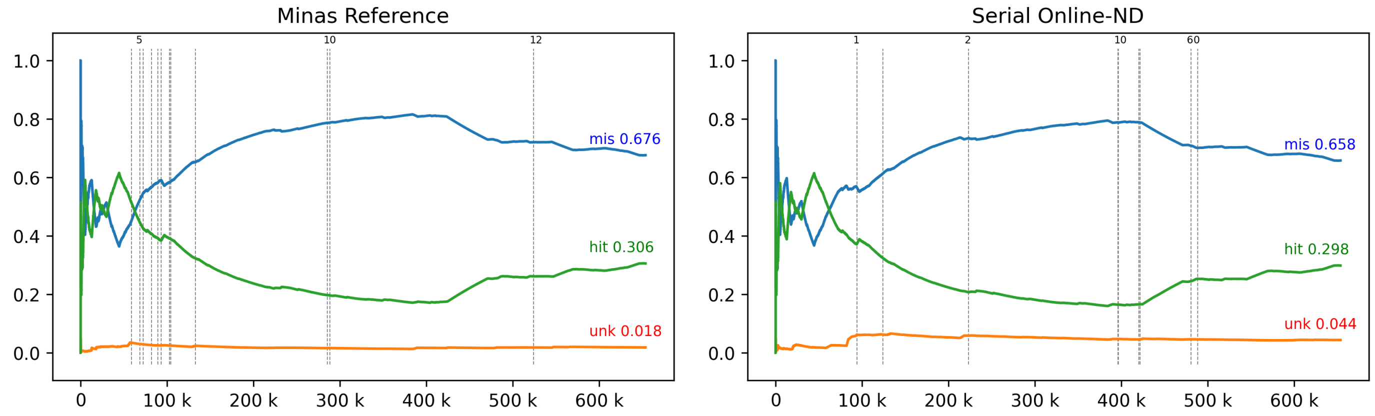
[4] K. A. da Costa, J. P. Papa, C. O. Lisboa, R. Munoz, and V. H. C. de Albuquerque, “Internet of Things: A survey on machine learning-based intrusion detection approaches,” *Computer Networks*, vol. 151, pp. 147–157, 2019.

[5] E. R. d. 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 2015. [Online]. Available: https://doi.org/10.1007/s10618-015-0433-y

[6] G. W. Cassales, H. Senger, E. R. DE FARIA, and A. Bifet, “IDSA-IoT: An Intrusion Detection System Architecture for IoT Networks,” in *2019 IEEE Symposium on Computers and Communications (ISCC)*, June 2019, pp. 1–7. [Online]. Available: https://ieeexplore.ieee.org/document/8969609/

[7] 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,” *Proceedings of the 1st Workshop on Building Analysis Datasets and Gathering Experience Returns for Security, BADGERS 2011*, pp. 29–36, 2011.

[8] E. R. de Faria, I. R. Goncalves, J. Gama, and A. C. P. d. L. F. Carvalho, “Evaluation of Multiclass Novelty Detection Algorithms for Data Streams,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 11, pp. 2961–2973, nov 2015. [Online]. Available: http://ieeexplore.ieee.org/document/7118190/



(a) Reference Implementation

(b) Serial Implementation

Fig. 3: Validation Comparison: Stream hits and novelties visualization

TABLE III: Reference implementation: Confusion Matrix and Qualitative Metrics

| Labels | - | N | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Classes | | | | | | | | | | | | | | |
| A | 3774 | 438750 | 123 | 145 | 368 | 8 | 52 | 165 | 1 | 1046 | 161 | 2489 | 71 | 26 |
| N | 8206 | 193030 | 0 | 79 | 44 | 0 | 0 | 0 | 229 | 181 | 154 | 4066 | 289 | 0 |
| Assigned | - | N | A | A | A | A | A | A | N | A | A | N | N | A |
| Hits | 0 | 193030 | 123 | 145 | 368 | 8 | 52 | 165 | 229 | 1046 | 161 | 4066 | 289 | 26 |

TABLE IV: Serial implementation: Confusion Matrix and Qualitative Metrics

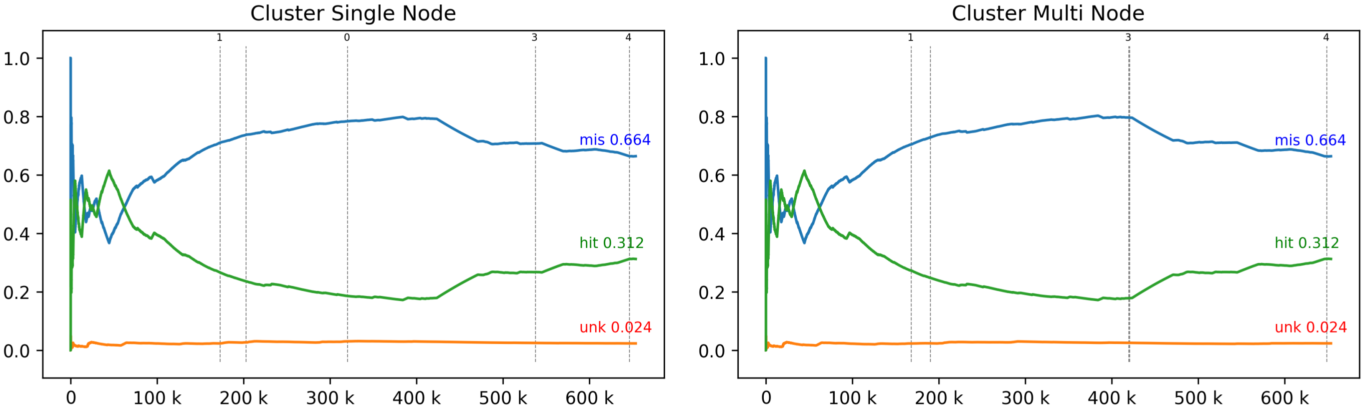
| Labels | - | N | 0 | 1 | 2 | 4 | 5 | 6 | 7 | 8 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Classes | | | | | | | | | | | |
| A | 16086 | 429765 | 94 | 995 | 104 | 0 | 23 | 3 | 29 | 46 | 34 |
| N | 12481 | 193642 | 3 | 94 | 0 | 47 | 0 | 0 | 0 | 11 | 0 |
| Assigned | - | N | A | A | A | N | A | A | A | A | A |
| Hits | 0 | 193642 | 94 | 995 | 104 | 47 | 23 | 3 | 29 | 46 | 34 |

TABLE V: Reference implementation: Confusion Matrix and Qualitative Metrics

| Labels | - | N | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|---|---|
| Classes | | | | | | | |
| A | 12282 | 433797 | 147 | 952 | 0 | 0 | 1 |
| N | 3088 | 203019 | 40 | 99 | 27 | 5 | 0 |
| Assigned | - | N | A | A | N | N | A |
| Hits | 0 | 203019 | 147 | 952 | 27 | 5 | 1 |

TABLE VI: Serial implementation: Confusion Matrix and Qualitative Metrics

| Labels | - | N | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|---|---|
| Classes | | | | | | | |
| A | 12282 | 433797 | 147 | 952 | 0 | 0 | 1 |
| N | 3088 | 203019 | 40 | 99 | 27 | 5 | 0 |
| Assigned | - | N | A | A | N | N | A |
| Hits | 0 | 203019 | 147 | 952 | 27 | 5 | 1 |



(a) Parallel single-node

(b) Parallel multi-node

Fig. 4: Parallelism Comparison: Stream hits and novelties visualization