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
Email: {luispuhl, gwcassales}@gmail.com, {hermes@, helio@dc.}ufscar.br

***Abstract*—The 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

The advent of Internet of Things (IoT) is growing the count and diversity of devices on edge networks, this growth increases network traffic patterns and extends opportunities for cyber attacks presenting new challenges for network administrators. To address those challenges new Network Intrusion Detection Systems (NIDS) and architectures can be explored, especially in Fog Computing and Data Stream (DS) areas.

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 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.

CNPq

II. Related Work

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

III. Proposal

Fog computing infrastructure aims to offload computing resources from cloud providers by placing edge devices closer to end-users and/or data sources.

Objective: Distributed novelty detection in streams using limited hardware.

Previous attempts to attain the objective of distributed fast

IV. Implementation

The original MINAS algorithm has a companion implementation (*Ref*) written in Java using MOA library base algorithms such as K-means and CluStream. *Ref* employs Java's double, a $64bits$ number whose precision is not absolutely necessary and, as it is often necessary to shuffle between nodes via network and a small economy could be made with only a float number with $32bits$. 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 [4].

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, optimality an unique item identifier (*uid*) can be provided. For output information and format the decision isn't so clear as we can't predict

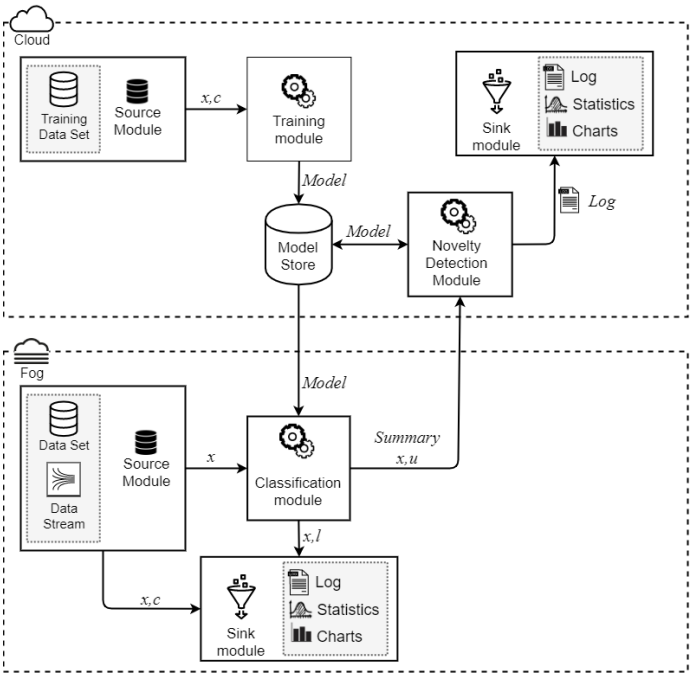

Fig. 1: *MFOG* architecture overview.

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 de accessed) meaning the format can be defined as a tuple containing *uid* and assigned label.

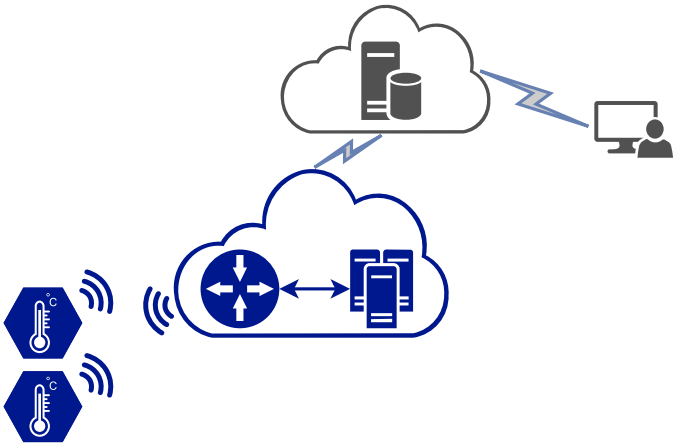

Fig. 2: *MFOG* physical architecture overview with cloud model storage. From bottom left to top right: sensor network; fog containing gateway router and novelty detection cluster; cloud storage for model, alarms and statistics and; human supervisor addressing alarms and statistics.

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 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 creation is shown in Figure 3.

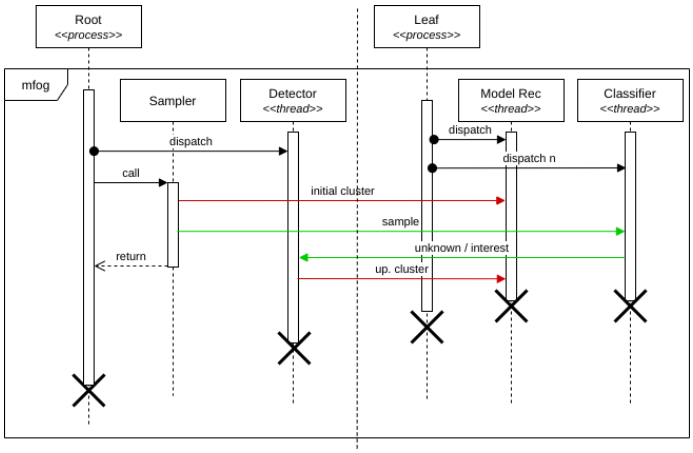

Fig. 3: *MFOG* life line overview.

Polices

- Detecção de novidades e manutenção de modelo em ambiente distribuído:
- Mecanismo de ND local (síncrono) vs nuvem quanto à atraso de definição de modelo (nesse ponto é onde a hipótese prevê maior diferença, grande ponto de interesse);
- Mecanismo de esquecimento local vs global (modelo único ou por nó);
- Atraso na reclassificação dos desconhecidos;

The Evaluation Module was also build following reference techniques like multi-class confusion matrix with label-class association [4] to extract classification quality metrics.

V. Experiments and Results

For the experimental setup we dedicated 3 Raspberry Pi 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) [5]. This segment was filtered (from 7865245 instances) to only examples associated to known attack types identified by existing IDS, and attack types with more than 10000 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 72000 instances for training set and 653457 for test set, containing 206278 N (normal) class and 447179 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.

BEGIN Oritações de leitura das métricas e visualizações.

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 [7] 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 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.

END Oritações de leitura das métricas e visualizações.

B. Results Discussion

Four main experiments need detailed discussion: (a) reference implementation of Minas (*Ref*) [4]; (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 I.

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.

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

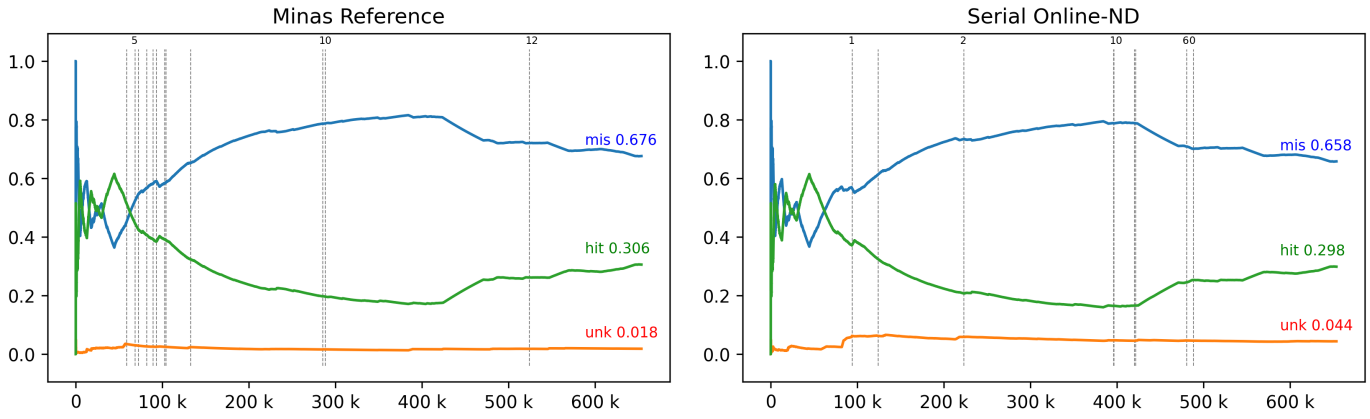

Fig. 4: Reference (a) and Serial (b) implementation: Stream hits and novelties visualization

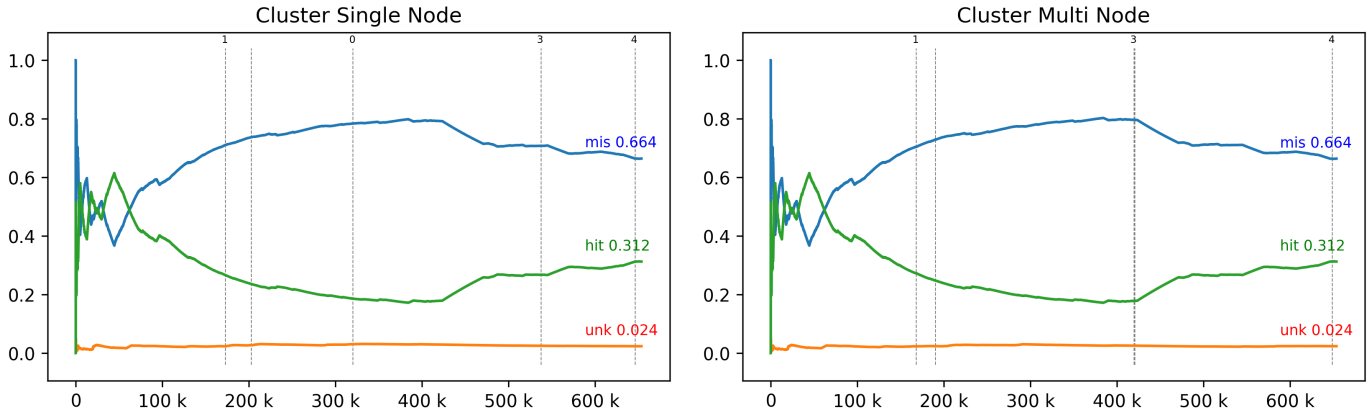

Fig. 5: Parallel single-node (c) and multi-node (d) modes: Stream hits and novelties visualization

TABLE I: 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

VI. Conclusion

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] 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

[5] 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.

[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] E. R. de Faria, I. R. Goncalves, J. ao 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/

TABLE II: 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 III: 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 IV: 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 V: 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