Implementing and Deploying an ML Pipeline for IoT Intrusion Detection with Node-RED

Yimin Zhang, Barikisu Asulba
Nuno Schumacher, Mario Sousa
Pedro Souto, Luis Almeida
{up202100775,up202103270}@edu.fe.up.pt
{nschumacher,msousa,pfs,lda}@fe.up.pt
CISTER / Fac. Eng. University of Porto
Porto, Portugal

Pedro M. Santos
pss@isep.ipp.pt
CISTER
School of Eng. Polytechnic of Porto
Porto, Portugal

Nuno Martins
Joana Sousa
nuno.mmartins@nos.pt
joana.sousa@nos.pt
NOS Inovação
Lisboa, Portugal

ABSTRACT

Edge devices in IoT ecosystems are subject to cyber-attacks (either as targets or participants), and the use of Machine Learning (ML) in said devices can facilitate intrusion detection locally, reducing the reliance on cloud infrastructure and increasing data privacy. This paper describes the implementation of an IoT-oriented application (use-case) that leverages ML on the edge, namely on the router deployed by an Internet Service Provider (ISP) at the customer premises, to detect potentially malicious traffic involving the customer's IoT nodes. We evaluate several middleware solutions regarding their support for ML applications in embedded devices, with a focus on low-code and event-driven approaches. We report the challenges and lessons learned in transferring an ML pipeline for intrusion detection, originally developed in a native Linux system, to a description in the selected middleware, Node-RED. Most of the processing itself is assured by the services of the original implementation, while Node-RED essentially acts as a control plane for coordinating those services. We also describe the deployment of the ML pipeline based on Node-RED on the edge device (router), and provide a characterization of the resulting solution.

CCS CONCEPTS

• **Software and its engineering → Embedded software**; • **Security and privacy → Intrusion detection systems**.

KEYWORDS

intrusion detection, IoT, machine learning, embedded systems, edge computing

ACM Reference Format:
Yimin Zhang, Barikisu Asulba, Nuno Schumacher, Mario Sousa, Pedro Souto, Luis Almeida, Pedro M. Santos, Nuno Martins, and Joana Sousa. 2023. Implementing and Deploying an ML Pipeline for IoT Intrusion Detection with Node-RED. In *Cyber-Physical Systems and Internet of Things Week 2023 (CPS-IoT Week Workshops '23), May 9–12, 2023, San Antonio, TX, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3576914.3589807

CPS-IoT Week Workshops '23, May 9–12, 2023, San Antonio, TX, USA

ACM ISBN 979-8-4007-0049-1/23/05…$15.00
https://doi.org/10.1145/3576914.3589807

1 INTRODUCTION

The reduced price and wide range of Internet-of-Things (IoT) products has resulted in the increased adoption of IoT devices in households. The exposure surface to cyber-attacks has therefore grown in proportion to IoT adoption, as IoT devices are often poorly secured and thus subject to zero-day exploits, justifying the use of cyber-security protection systems.

Security attacks are constantly evolving, and new IoT devices are constantly coming to market. A cyber-security protection system should ideally be able to handle types of network traffic it has not yet come across, and be able to adapt to different legitimate Internet usage patterns in distinct households. Machine Learning (ML) comes into play through its ability to learn network traffic patterns that are legitimate, and to apply that knowledge to identify anomalous patterns that may indicate malicious traffic. Compared to rule based Intrusion Detection Systems(IDSs), ML based IDSs have the potential to identify unseen attacks [1].

One way to implement this protection system is to resort to the Cloud, taking advantage of its wealth of resources to run complex ML models and potentially their online training. This approach implies sending traffic features over public networks to the Cloud, which further widens the exposure surface.

An alternative approach is to run the protection system at the Edge of the Internet, in the household gateway/router. The motivation is that the Customer Premises Equipment (CPE) deployed by ISP can, due to its strategic position, play a pivotal role in increasing the security of the customer's IoT ecosystem in two ways, doing a quicker detection of potentially malicious traffic and confining the exposure of private customer data.

Exploring this Edge-based approach to intrusion detection in IoT home ecosystems is the target of one of the five use cases considered in the MIRAI project [10] and it is the focus of this paper. In particular, we address the following questions. Which kind of ML models are suitable? How can the deployment and operation of ML pipelines be streamlined in a way that boosts reconfigurability? How well can the pipeline perform embedded in the CPE given its resource constraints? This paper reports ongoing work to answer the questions above and presents early results achieved with a real CPE.

An ML pipeline capable of locally assessing in real-time the maliciousness of network traffic was initially developed in a standard Linux installation using Python and scientific libraries, system commands (`tail`), and external tools (`Tstat`). We briefly describe

this pipeline and mention the results obtained in Section 2, when executed on a powerful desktop computer.

In order to allow the pipeline to be rapidly reconfigured, Section 3 evaluates several possible middlewares for supporting, launching, and coordinating the execution of the ML pipeline embedded on the CPE, acting as an edge device. We further report in Section 4 the challenges identified in porting the pipeline to an implementation based on Node-RED [12] and its deployment using one particular ML model. We also show preliminary results of memory footprint and execution time of the ML model in the CPE in Section 4.1. Finally we draw the conclusions of the work reported herein, highlighting the current directions of the work in progress.

2 MACHINE LEARNING PIPELINE

Given the advances in computing power, ML is becoming ubiquitous, penetrating many domains of our lives to learn empirically from virtually any kind of process data. The IoT domain is no exception and a myriad of works are available in the literature combining IoT and ML. For example, the recent survey in [8] refers to many cutting-edge ML methods used in heterogeneous IoT environments. In our case, we are interested in using ML for intrusion detection in IoT ecosystems. We focus on analyzing TCP flows since the data available in public datasets is highly biased to this type of traffic. However, UDP communications can also be analyzed upon a small adaptation of the traffic features used.

2.1 ML Models

One difficulty of intrusion detection, particularly in IoT ecosystems, is the low availability of malicious datasets for training the ML models. One way to circumvent this difficulty consists in using the so-called One-Class models, which are trained with traffic of one class, only, and detect traffic that falls outside that class. This is adequate to intrusion detection since we can train the models with legitimate traffic, only, and any traffic that rests outside this class is marked as potentially malicious.

With this in mind, we considered several different One-Class ML models, namely:

- One-Class Support Vector Machine (OCSVM)
- Elliptic Envelope (EE)
- Isolation Forest (IF)
- Local Outlier Factor (LOF)

These models have different internal structures, leading to different model sizes, execution times and detection metrics (particularly accuracy and detection rate). A thorough discussion of these models can be found in [17].

2.2 Pipeline Description

To carry out the desired intrusion detection at the CPE, we developed the ML pipeline shown in Figure 1 [17]. The main development of the pipeline was done in Python on a Linux machine, particularly using scientific-oriented libraries such as *numpy*, *pandas*, and *scikit-learn*.

The pipeline performs solely the prediction (inference) stage of the ML models utilization. The pipeline is composed of three parts: (i) data collection; (ii) pre-processing; and (iii) prediction. The used ML models were trained offline.

Data collection: This is conducted using tools `tcpdump` [1], a tool to capture network packets from a selected interface, and `Tstat`[2], a tool to extract flows from those collected packet traces. Both `tcpdump` and `Tstat` are constantly running.

Pre-processing: The output of `Tstat` is a list of identified flows and its description according to a set of characteristics. The prediction pipeline expects a set of features that describe the flow (currently 31 are used among counts of packets and flags, and response times). A noteworthy limitation is that we cannot read the output of `Tstat` directly via `pipe`; as such the output of `Tstat` is written to a file that a system command (`tail`) monitors for changes.

Prediction: This step is where the models are actually executed. Our pipeline is designed to allow the parallel execution of multiple models, using a decider module to combine together the respective individual classifications into a single output. This parallel execution allows exploiting the heterogeneity of the referred models, resulting in improved detection performance. The models were also implemented in Python.

2.3 Performance of the ML Pipeline

This pipeline was initially tested [17] on an Ubuntu machine with 8GB of RAM and a 4-core CPU Intel(R) Core(TM) i5-4570 with a clock speed of 3.20 GHz and a nominal clock-rate of 1.80GHz. These initial tests used statically configured Python scripts instead of a programming framework such as those analysed in this paper.

The models, described in Section 2.1, were trained with 2000 samples each and were subject to parameter tuning with the same number of validation samples, and their decisions aggregated by majority voter.

The combination of the models, after parameter tuning, achieved an accuracy of around 90% for legitimate traffic in general, and 99% when this traffic was captured in same network conditions as training. It also detected around 95% of various types of attacks. A complete performance characterization can be found in [17].

3 SELECTED PROGRAMMING FRAMEWORKS

When considering the deployment of the referred ML pipeline in an embedded system with limited resources, such as the CPE, it is important to design it in a way that supports modularity and enhances reconfigurability. This allows an easy addition or replacement of ML models, e.g., upon retraining, of feature extraction tools, pre-processing methods or prediction aggregation methods. The direct implementation in Python on Linux does not meet this level of modularity, which prompted us to explore other programming frameworks that do.

A wide range of frameworks were initially considered, but the list was cut down to three for their ability to support flow-based applications as well as their popularity, which increases the likelihood of continued future maintenance and support. The following chosen frameworks were more closely evaluated: Node-RED [12], GStreamer [6] and Arrowhead [2]. The first two are design frameworks while the latter is a full middleware layer.

[1] https://www.tcpdump.org/
[2] http://tstat.polito.it/

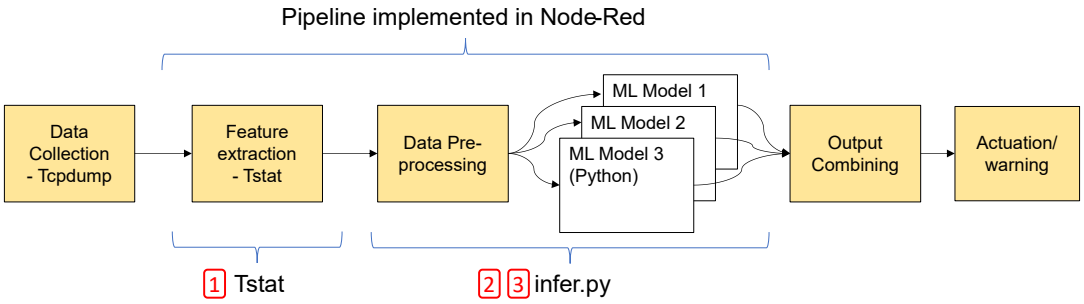

Figure 1: ML Pipeline for Prediction. Bottom brackets and boxed numbers identify components of Node-RED flow (Section 4).

In this section we introduce these frameworks, we discuss their applicability to the use case at hand and we present a brief qualitative comparison to support the choice for one, namely Node-RED.

3.1 Node-RED

Node-RED [12] is a flow-based programming tool, originally developed by IBM's Emerging Technology Services team, for wiring together hardware devices, APIs and online services. Currently, Node-RED is maintained and developed by the OpenJS Foundation, being open source and providing a rich library of *nodes* and *flows*.

The term *nodes* refers to functional blocks based on JavaScript and Node.js. Developers construct their applications by dragging and wiring existing or newly created nodes on a web-based graphical editor, making it a low-code platform. Being built on Node.js, Node-RED is well suited to run at the edge of the Internet on low-cost hardware, e.g., a Raspberry Pi, as well as in the cloud.

From version 2.10.0 onwards Node-RED provides a set of nodes to quickly create a live data dashboard. After installing the widget nodes like other Node-RED nodes, it allows showing charts, gauges, sliders, etc, to visualize the data and control the process.

3.1.1 Node-RED and Machine Learning. Given the wide-spread use of both Node-RED and ML algorithms, many previous projects have integrated both. The Node-RED library includes an ML module, namely *node-red-contrib-machine-learning*, that is based on the Python ML libraries SciKit-Learn and Tensorflow, and includes 9 classifier nodes (decision tree, deep neural network, k−neighbors, etc.). However, it has not been updated in the last 5 years, and some issues related to its use remain unsolved. Other alternative nodes provided by other contributors to the library also exist, e.g., *node-red-contrib-tensorflow*, *node-red-contrib-facial-recognition*.

An alternative is using TensorFlow.js [19] directly, a JavaScript ML library that can run in the browser or in Node.js. IBM proposes using this library for building ML applications on Node-RED [7].

Previously published work related to running ML applications on Node-RED is still scarce. One such work describes using Node-RED to enable design of custom architectures of Generative Adversarial Networks (GAN). Raunak Sinha et al. [18] propose a GAN authoring system, *AuthorGAN*, based on six different reusable components/modules ((i) real training data, (ii) generator, (iii) discriminator, (iv) loss function, (v) optimizer function and (vi) training process). Node-RED is used for its intuitive visual interface that allows designing and building GAN models from scratch without the need for writing code. This work is similar to ours in that both use Node-RED as an *orchestration* framework used to combine different model components.

3.2 GStreamer

GStreamer is an open source and multi-platform framework adapted to building applications that process and transform data-flows. A GStreamer application consists of several *pipelines*. The pipeline can be considered a representation of a particular data transformation workflow and is represented by a chain/graph of nodes/*elements*. Elements performing related functions are often organized together in a dynamic loadable module called a *plugin*.

Although the framework itself is not restricted to media, a large proportion of the available *elements* perform operations on media data types (e.g., handling of container formats, network streaming, encoding/decoding). GStreamer can however incorporate new plugin modules for constructing graphs of non-media-handling components.

Since GStreamer supports all major operating systems and architectures, it is suitable for a wide range of systems, from edge devices to supercomputers.

3.2.1 GStreamer and Machine Learning. NVIDIA provides a GStreamer toolkit, namely *DeepStream* SDK [14], for AI-based multi-sensor processing, video, audio, and image understanding. DeepStream is supported on platforms that contain an NVIDIA® Jetson™ module [13], i.e., one of a series of GPU-based boards designed for accelerating machine learning applications. DeepStream supports C/C++, Python and Graph Composer, a low-code development tool developed by NVIDIA used to construct pipelines with drag-and-drop operations.

Ricardo et al. [16] built an end-to-end UAV streaming system with low latency based on Jetson and GStreamer. The platform is used to authenticate each UAV and to establish a link to a WebRTC-enabled media gateway server so clients can access the media through a Website. In [9], Kleiner et al. present a prototype software implementation of the flux tensor motion flow algorithm based on the GStreamer multimedia framework. By leveraging the computational performance of NVIDIA GPUs and CUDA platform, real-time motion detection is achieved in ultra-high definition video streams. Goel et al. [5] demonstrate a system called *CaptionAI* used for speech to text transcription, multilingual translation, and real-time closed captioning. Gstreamer is used for live conversion of audio

Table 1: Comparison of Node-RED, GStreamer, Arrowhead

Features	Node-RED	GStreamer	Arrowhead
Distribution	Distributed	Distributed	Centralized
Open Source	Yes	Yes	Yes
Low-code	Yes	No	No
Light-weight	Yes	Yes	No
Real-time	No	Yes	Yes
Run-time	Node.js	C run-time	Java Runtime Environment (JRE)
Language	JavaScript	C	Java
Required Device Size	Small to large	Small to large	Small to large
Supporters	OpenJS Foundation	GStreamer	Arrowhead
Tutorials/Documentation	Yes	Yes	Yes
Communication Protocols	HTTP, MQTT, Modbus	RTP, RTSP	HTTP, CoAP, MQTT, OPC-UA
Message Patterns	Pub/Sub	Pub/Sub	Req/Repl, Pub/Sub
Parallel	Yes	Yes	Yes
Robust	Yes	Yes	Yes
Easy to deploy	Easy	Hard	Medium
Dynamic Service Binding	No	No	Yes

signal, and to send the audio file to an automatic speech recognition server. The speech recognition server utilizes Deep Neural Network (DNN) to train the acoustic model. In [3] Brewer et al. use GStreamer to create benchmarks on a supercomputer. GStreamer is responsible for streaming both synthetic and real Motion Imagery (MI).

3.3 Arrowhead

The Arrowhead framework [2] provides an architecture for building IoT-based (mostly Industrial) Automation systems. It is based on a service-oriented architecture (SOA) [4] and features a local cloud.

To establish an Arrowhead framework on the local cloud, three mandatory core systems are required:

- The Service Registry System
- The Authorization System
- The Orchestration System

These core systems, together with at least one application system, form a closed group of systems with physical proximity to the edge. At the local cloud level, a private network connects the involved systems. Inter cloud service exchanges are facilitated by two additional core systems: a gateway and gatekeeper. Using this framework it is possible to add more core systems so as to extend the basic functionality, including DataManager, QoS, EventHandler, and Configuration.

3.3.1 Arrowhead and Machine Learning. Published work related to Arrowhead and ML is also rare. In [15], H. Pettinen and D. Hästbacka reveal that Arrowhead can support use cases requiring soft real-time models. When performing inference tasks in ML, services could benefit from Arrowhead to ensure timing properties and quality of service.

In [11], Nilsson et al. tackle the interoperability issues arising when building dynamic IoT solutions. Although this paper is not about applying ML in Arrowhead, it provides a view of decomposing problems to ML tasks in Arrowhead.

3.4 Comparison

Table 1 presents a comparison between the three frameworks. Although each framework has its own advantages, Node-RED was the most promising for our requirements.

The Arrowhead framework was not considered further because it requires several core services to execute concurrently with the application itself. The registration and orchestration services support dynamic on-line reconfiguration of the application, a feature not present in the other frameworks, but not needed for our use case. Running all core services in the CPE could quickly become problematic due to its limited memory resources. Another option would be to run the core services directly in the cloud. This alternative was quickly discarded due to the very large number (hundreds of thousands) of CPEs that it would have to orchestrate.

The other two frameworks are both sufficiently light-weight for our purposes. Of these two, Gstreamer presented a more cumbersome API, and therefore more difficult to use and integrate into our application.

An important requirement in our application is the possibility of re-configuring the pipeline. Both GStreamer and Node-RED support rapid off-line reconfiguration of the structure of an application (i.e. re-configuring the ML pipeline), but once again Node-RED is much easier to use with its intuitive graphical interface.

In our scenario the re-configured pipeline must then be propagated to potentially hundreds of thousands of CPEs deployed in customer premises. Node-RED comes out ahead in this aspect as well as the Node-RED run-time that co-ordinates the execution of Node-RED applications comes with a network facing API through which new applications may be deployed, or existing applications stopped. Doing the same with GStreamer requires either implementing an equivalent re-configuration server to run on each CPE, or making do with existing standard Linux tools (for example, remote execution of command line programs).

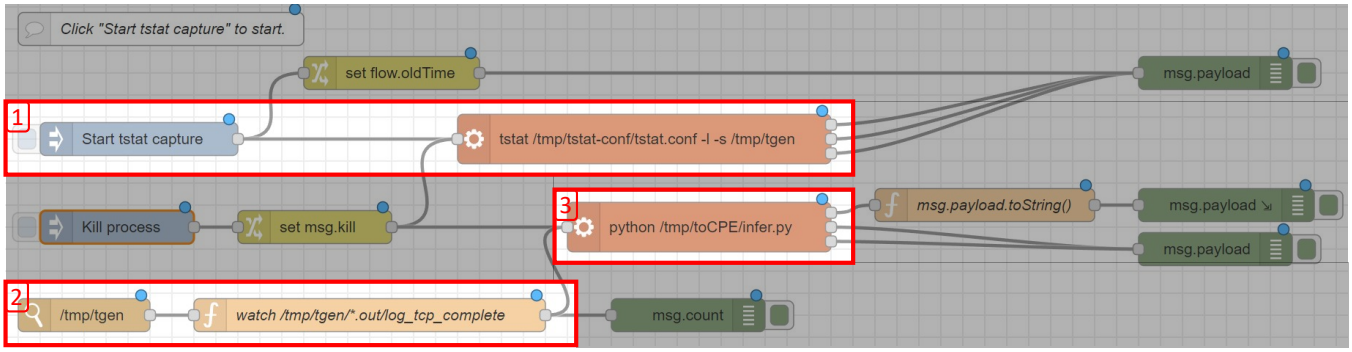

Figure 2: ML Pipeline in the NodeRed Web Editor

Out of the three frameworks considered, Node-RED is the only that does not have explicit support for some level of timing guarantees. Even though our application processes traffic data in real-time, timing requirements are relatively relaxed given that detected IoT flows are marked as potentially malicious and further actions are currently taken by a human operator.

4 PIPELINE DEPLOYMENT IN NODE-RED

We report the challenges we found when transcribing the ML pipeline described in Section 2 to a Node-RED implementation. Figure 2 shows the produced flow in the Node-RED web editor. Note that the pipeline is not as straightforward as the one shown in Figure 1. Multiple reasons concur to this. For convenience in the following discussion, some sub-flows (*chains*) in Figure 2 are isolated and numbered; the numbers can be connected to those that are also shown in Figure 1.

Node-RED is event-driven, but some of the services external to Node-RED will execute permanently. External services are implemented using the `exec` node (in red). It is the case of `Tstat` and chain 1 in Figure 2 corresponds to the initialization of `Tstat`. The *inject* node on the left of the chain (blue) provides a single input to the *exec* node at the start of the operation.

Another notable aspect is that the output of `Tstat` cannot be read directly via `pipe`. This issue led to the need of a second flows that needs to be initiated in parallel. Chain 2, initiated by the `watch` node (a node that watches for changes on files or folders), will monitor those files and, when a new entry is detected, it triggers the execution of the pre-processing and prediction Python scripts.

Finally, chain 3 corresponds to the actual inference using a Python script. This is only run when a new sample is reported by chain *2*. Ultimately, it produces a classification that is presented to the user.

4.1 Initial Evaluation on Embedded System

We deployed the Node-RED pipeline in an embedded system provided by an industrial partner, notably an Internet Service Provider (ISP). The target embedded system is a fully operational gateway and router at the premises of a test customer, processing actual Internet traffic. It is a dual-core Atom(TM) CPU CE2752 @ 2.00GHz with 640MB of RAM.

Due to limitations of the target platform, we do not currently deploy all the developed ML models. From those referred in Section 2, we deployed just one, the Elliptic Envelope. The reason relates to the Python dependencies required to run most of the models, that the target platform did not support up to the required version. Supporting more models is on-going work.

We evaluate the performance of the Node-RED pipeline in the CPE embedded system in terms of **memory footprint** and **response time**. It should be noted that performance of network traffic routing is not expected to be significantly affected by the addition of the Node-RED pipeline. This is due to the CPE's architecture - the main CPU merely handles the first few packets of each stream and off-loads routing of subsequent packets to dedicated routing hardware.

The memory footprint of Node-RED piepline is reported in Table 2. The main cost comes from the space taken to hold the Node-RED executable (42.6MB) and Python's library `numpy` (14.1MB), which required a judicious management of the runtime support installed in the CPE. On the other hand, the ML model we deployed took just a marginal amount of memory, as well as the Python script that uses it (`infer.py`). However, this is rather variable between models and as to be assessed in each case.

Component	Size
Node-RED installation	42.6 MB
Elliptic Envelope Model (.ml)	174.6 KB
Python Scripts (infer.py)	7.8 KB
Python dependencies	14.2MB

Table 2: Memory footprint of Node-RED implementation.

In what concerns the response time, we took two measurements of the pipeline, namely the time taken by the prediction stage and the time taken by the pre-processing and prediction stages together. The histograms of the values observed during approx. 10 minutes of operation are shown in Figures 3 and 4, respectively.

A summary of the values is reported in Table 3. The average value of the predictor response time is approx. 4.6ms while the pre-processing stage takes an additional 1ms on average. However, rare longer response times were observed (approx. 27ms), most likely caused by surges in both network load and computational load in the CPE.

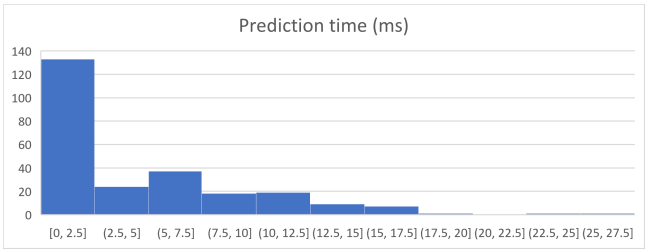

Figure 3: Response time of prediction module (ms)

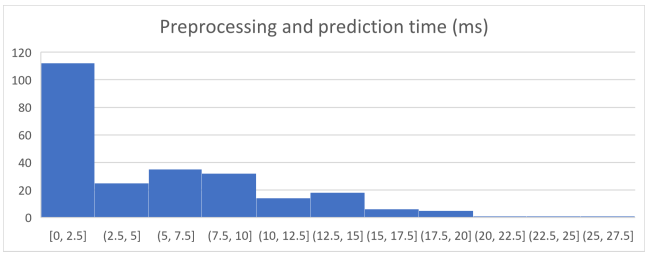

Figure 4: Response time of preprocessing + prediction (ms)

	Pred (ms)	Pre + Pred (ms)
Min	0.95	1.23
Max	27.07	27.35
Median	1.39	3.61
Avg	4.61	5.60

Table 3: Response time.

Two aspects still need a special mention. These response times were obtained from a real device trace in full operation. The response times reported are just a part of the total response time from the input of a new flow and the corresponding classification. This requires a detailed profiling of `tstat` response time, which is currently ongoing. Nevertheless, the partial response times we observed already provide evidence that the desired detection of potentially malicious IoT traffic can be done in the sub 100ms range.

5 CONCLUSION

This paper addressed the implementation of an intrusion detection system for an IoT home ecosystem in the edge, namely in the home router of the respective ISP, a.k.a. CPE. This implementation requires a high degree of reconfigurability and for this reason we compared three industrial implementation frameworks, selected for their suitability to describe and run execution pipelines. The choice fell on Node-RED for its high modularity and low-code approach. Then, the paper discussed the difficulties that emerged when deploying the Node-RED based ML pipeline in the CPE. We identify the limitations of external tools and the solutions that were found to integrate those in the Node-RED description of the pipeline. Finally we report early results of memory footprint and response time of the ML pipeline that confirm the feasibility of the Node-RED based approach.

Future work includes carrying out quantitative comparisons among the frameworks and between different deployments. We will also extend the pipeline with more models and do a full characterization of all services involved, as well as addressing the scalability of the approach to allow handling larger loads in parallel, e.g., through the use of multi-processors or of a distributed approach.

ACKNOWLEDGMENTS

This work was partially supported by National Funds through FCT/MCTES (Portuguese Foundation for Science and Technology), within the CISTER Research Unit (UIDB/04234/2020), and by the Portuguese National Innovation Agency (ANI) through the Operational Competitiveness Programme and Internationalization (COMPETE 2020) under the PT2020 Partnership Agreement, through the European Regional Development Fund (ERDF), within project(s) grant nr. 69522, POCI-01-0247-FEDER-069522 (MIRAI).

REFERENCES

[1] Mouhammd Alkasassbeh and Sherenaz Al-Haj Baddar. 2022. Intrusion Detection Systems: A State-of-the-Art Taxonomy and Survey. *Arabian Journal for Science and Engineering* (2022), 1–44.
[2] Arrowhead. 2016. *Arrowhead Framework Wiki.* http://www.arrowheadproject.eu/arrowhead-wiki/
[3] Wesley Brewer, Chris Geyer, Dardo Kleiner, and Connor Horne. 2021. Streaming Detection and Classification Performance of a POWER9 Edge Supercomputer. In *2021 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, Virtual, 1–7. https://doi.org/10.1109/HPEC49654.2021.9622852
[4] Jerker Delsing. 2017. *Iot automation: Arrowhead framework.* Crc Press, Boca Raton, FL.
[5] Nagendra Kumar Goel, Mousmita Sarma, Saikiran Valluri, Dharmeshkumar Agrawal, Steve Braich, Tejendra Singh Kuswah, Zikra Iqbal, Surbhi Chauhan, and Raj Karbar. 2019. CaptionAI: A Real-Time Multilingual Captioning Application.. In *INTERSPEECH*. ISCA, Graz, 4632–4633.
[6] GStreamer 2023. *The GStreamer Website.* https://gstreamer.freedesktop.org/
[7] IBM. 2020. Build a machine learning node for Node-RED using TensorFlow.js. https://developer.ibm.com/tutorials/building-a-machine-learning-node-for-node-red-using-tensorflowjs/
[8] Ahmed Imteaj, Urmish Thakker, Shiqiang Wang, Jian Li, and M. Hadi Amini. 2022. A Survey on Federated Learning for Resource-Constrained IoT Devices. *IEEE Internet of Things Journal* 9, 1 (2022), 1–24. https://doi.org/10.1109/JIOT.2021.3095077
[9] Dardo D Kleiner, Kannappan Palaniappan, and Gunasekaran Seetharaman. 2016. Stream implementation of the flux tensor motion flow algorithm using GStreamer and CUDA. In *2016 IEEE Applied Imagery Pattern Recognition Workshop (AIPR)*. IEEE, Washington, DC, 1–7.
[10] ITEA 2020. *MIRAI Project.* ITEA. https://project-mirai.eu/
[11] Jacob Nilsson, Fredrik Sandin, and Jerker Delsing. 2019. Interoperability and machine-to-machine translation model with mappings to machine learning tasks. In *2019 IEEE 17th International Conference on Industrial Informatics (INDIN)*, Vol. 1. IEEE, Helsinki-Espoo, 284–289.
[12] Node-RED. 2020. *The Node-RED Website.* https://nodered.org/
[13] NVIDIA. 2022. *Embedded Systems with Jetson.* https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/
[14] NVIDIA. 2022. *NVIDIA DeepStream SDK.* https://developer.nvidia.com/deepstream-sdk
[15] H. Pettinen and D. Hästbacka. 2022. Service orchestration for object detection on edge and cloud in dependable industrial vehicles. *Journal of Mobile Multimedia* (2022), 1–26.
[16] Ricardo Sacoto-Martins, João Madeira, J. P. Matos-Carvalho, Fábio Azevedo, and Luís M. Campos. 2020. Multi-purpose Low Latency Streaming Using Unmanned Aerial Vehicles. In *2020 12th International Symposium on Communication Systems, Networks and Digital Signal Processing (CSNDSP)*. IEEE, Porto, 1–6. https://doi.org/10.1109/CSNDSP49049.2020.9249562
[17] Nuno Schumacher. 2022. *Anomaly detection models for cloud-edge intrusion detection in customer networks.* Ph. D. Dissertation. Universidade do Porto. https://hdl.handle.net/10216/142591
[18] Raunak Sinha, Anush Sankaran, Mayank Vatsa, and Richa Singh. 2019. AuthorGAN: Improving GAN Reproducibility using a Modular GAN Framework. arXiv:1911.13250
[19] TensorFlow. 2022. *TensorFlow.js.* https://www.tensorflow.org/js