Scheduling to the Rescue; Improving ML-Based Intrusion Detection for IoT

Scheduling to the Rescue; Improving ML-Based Intrusion Detection for IoT

Aria Mirzai*
aria.mirzai@ri.se
Dependable Transport Systems
RISE Research Institutes of Sweden
Borås, Sweden

Ali Zülfükar Coban*
zulfukar@chalmers.se
Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden

Magnus Almgren
magnus.almgren@chalmers.se
Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden

Wissam Aoudi
wissam.aoudi@clavister.com
Clavister
Gothenburg, Sweden

Tobias Bertilsson
tobias.bertilsson@clavister.com
Clavister
Gothenburg, Sweden

Abstract

With their inherent convenience factor, Internet of Things (IoT) devices have exploded in numbers during the last decade, but at the cost of security. Machine learning (ML) based intrusion detection systems (IDS) are increasingly proving necessary tools for attack detection, but requirements such as extensive data collection and model training make these systems computationally heavy for resource-limited IoT hardware. This paper's main contribution to the cyber security research field is a demonstration of how a dynamic user-level scheduler can improve the performance of IDS suited for lightweight and data-driven ML algorithms towards IoT. The dynamic user-level scheduler allows for more advanced computations, not intended to be executed on resource-limited IoT units, by enabling parallel model retraining locally on the IoT device without halting the IDS. It eliminates the need for any cloud resources as computations are kept locally at the edge. The experiments showed that the dynamic user-level scheduler provides several advantages compared to a previously developed baseline system. Mainly by substantially increasing the system's throughput, which reduces the time until attacks are detected, as well as dynamically allocating resources based on attack suspicion.

CCS Concepts: • **Security and privacy → Intrusion detection systems**; • **Computer systems organization → Processors and memory architectures**; • **Computing methodologies** → *Machine learning*; *Parallel computing methodologies.*

Keywords: Internet of things, Anomaly-based intrusion detection system, User-level scheduling, Model training

ACM Reference Format:
Aria Mirzai, Ali Zülfükar Coban, Magnus Almgren, Wissam Aoudi, and Tobias Bertilsson. 2023. Scheduling to the Rescue; Improving ML-Based Intrusion Detection for IoT. In *16th European Workshop on System Security (EUROSEC '23), May 8, 2023, Rome, Italy*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3578357.3589460

*Both authors contributed equally to this research.

EUROSEC '23, May 8, 2023, Rome, Italy

ACM ISBN 979-8-4007-0085-9/23/05…$15.00
https://doi.org/10.1145/3578357.3589460

1 INTRODUCTION

The Internet of Things signifies a major paradigm shift, that allows for the collection of information through connected devices but also poses security concerns [9] [15]. Such was the case with the Mirai Botnet [5], where a network of remotely controlled bots was used to launch Distributed Denial of Service (DDoS) attacks against the Domain Name System provider Dyn [2].

Therefore, early identification and reaction is crucial to stop an adversary from compromising devices and cause severe damage [4]. To accomplish this, IDS that monitor and analyse event streams have become invaluable tools. Anomaly-based IDS specifically have been increasingly used since it allows for monitoring event streams by deploying machine learning [8], [6].

However, ML models require extensive data collection as well as needing to be trained. Both of these are computationally heavy processes and require simultaneous handling, posing a great challenge for resource-limited IoT hardware. Remote offloading is not always a viable option either, due to restrictions on transferring sensitive data or simply a lack of network access. Additionally, monitoring multiple nodes (dealing with different event streams) requires an identical number of detection (or ML) algorithm instances to be executed in parallel.

The main contribution of this paper is investigating how a dynamic user-level scheduler can satisfy the following exigencies:

- Scalable performance improvement in accordance with the available CPU and memory resources.
- Enabling on-device retraining of an ML model without compromising other ongoing IDS tasks.
- Agnostic towards a class of ML algorithms, specifically data-driven and lightweight ones that work for machine-to-machine communication and monitor sensor data.
- Dynamic resource allocation based on attack suspicion.

Motivating examples for our work can be found in [8] and [17]. Both [8] and [17] state that the major bottleneck for ML-based IDS is the hardware limitations of IoT devices, such as computational power and limited memory. These restrictions result in lightweight IDS with only minimal security measures. Thus, a better support system capable of utilising the IoT hardware will allow for more advanced IDS and will be essential for the cybersecurity research field in IoT security. This is the main purpose of this work which explores performance improvements through a dynamic user-level scheduler.

Architecture designing for ML-based systems has become a broad research area [10], it is therefore important to emphasise that we are only investigating a minor problem within this domain. Note that we treat the detection algorithm as a black box, where our only concern is the amount of resources required for execution, resulting in a more algorithm-agnostic solution with broader support.

2 BACKGROUND

2.1 Parallelising Sequential Programs

Parallelisation of computer programs deals with utilising the hardware architecture to its maximum capacity. Simply put it is the concept of making sure processor cores can compute independent tasks in parallel, minimising both idleness and communication among cores [14].

To determine the independent tasks, the program is partitioned through a process of decomposing computations into smaller ones while data dependencies are identified. Data parallelism may be applied on an operation executed multiple times on data-structure elements (e.g., array elements), where each element is independent. This holds relevancy as we assume zero data dependencies between each instance of the detection algorithm [14].

2.2 Task Scheduling

Scheduling [14] is the assignment of tasks to processes or threads. A scheduling algorithm can either be static or dynamic. Static scheduling maps the order of task execution before program execution, whereas dynamic scheduling maps during run-time. Operating systems arrive with a generally purposed scheduler, but constructing one optimally tailored for intrusion detection tasks is a powerful approach for this work. There are three critical factors to consider when scheduling tasks; load balancing, information/data exchange between cores, and memory accesses.

The workload is evenly split between CPU cores in an effectively parallelised program. The tasks assigned to a CPU core should involve minimal communication between other cores since information exchange results in increased execution time.

Additionally, optimising the tasks for cache locality means that needed data is located close to the CPU cache, resulting in less memory access overhead and reduced execution time.

3 RELATED WORK

Offloading computation to the cloud has become an attractive solution to improve IoT performance. Several proposals by researchers which focus heavily on efficient resource management through task scheduling have been made for cloud computing. In [12], user requests (from the edge layer) are handled by a "broker" at the fog layer. The broker performs task scheduling by managing the computational resources of cloud servers and nodes at the fog level (here, each node is connected to devices at the edge layer). It calculates a task graph, a processor graph and a priority for each task before determining which cloud resource should execute the request.

HUNTER [16], and its improved version HunterPlus [7], focus on performing task scheduling by making use of AI at the fog layer as well, for efficient utilisation of cloud resources. They aim to handle user requests from the edge layer using AI-based task scheduling to effectively manage the energy usage of cloud computing and reduce operating costs for cloud providers.

The main limitation with these proposals, from a security perspective, is that remotely offloading tasks to the cloud may not be a viable option due to restrictions on transferring sensitive data or simply a lack of network access. Moreover, keeping the computation locally greatly increases security as this would decrease the communication and data transfer across layers for analysis. Consequently, the attack options for adversaries would be reduced.

PASAD [3] is a lightweight data-driven and anomaly-based IDS based on ML that monitors time series of sensor measurements in real-time. It is capable of executing reasonably fast since anomaly detection is made through a single matrix multiplication. Midbro [1] deploys PASAD in a real-world environment at an operational paper factory for 75 days to secure industrial control systems. In this work, we use PASAD as the detection algorithm while viewing it as a black box to demonstrate the capabilities of our scheduling solution with a ML algorithm on IoT hardware.

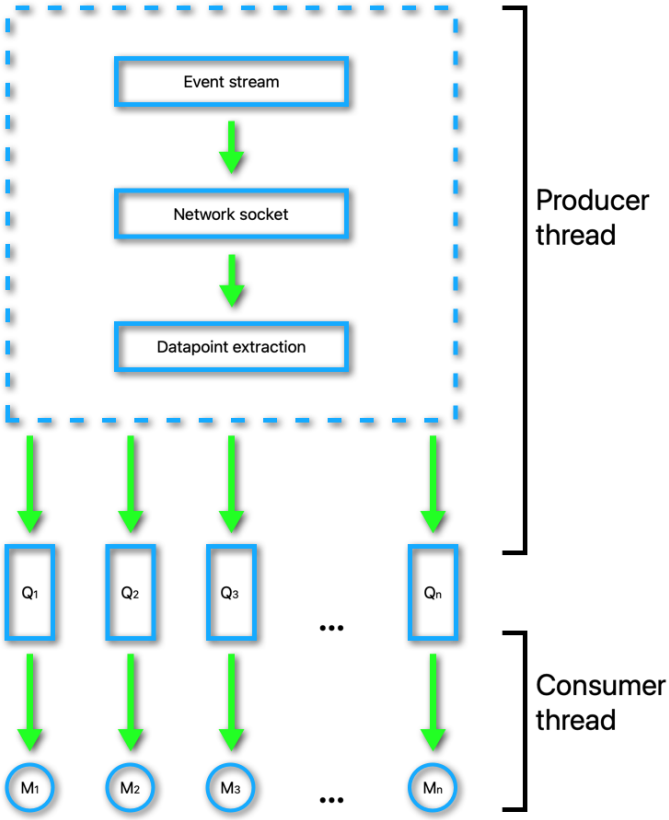

Figure 1. Overview of the baseline architecture where $M_1, .., M_n$ is the detection models and Q_n are the numbers of buffers. Note that each instance of the detection algorithm has its own unique buffer.

4 ARCHITECTURE DESIGN

4.1 The Baseline system

Existing IDS contain numerous extra features that are not relevant to this work. Therefore, constructing our own baseline architecture, only containing the essential features, allows for a better focus on the promised contributions. The baseline system partly acts as an experiment platform intended for analysing the impact of later changes comparatively. An overview of the system is provided in Figure 1.

4.1.1 The Producer. The producer thread is responsible for opening a socket to capture communication traffic, extracting particular data points from packet headers (payload is discarded), and then inserting these into one of the buffers. In case a buffer gets full, the oldest packets are overwritten, following the Midbro component [1] approach.

4.1.2 The Consumer. Parallel to the producer thread, the consumer thread pops elements from the buffers and feeds them to their respective model to execute anomaly detection. This occurs in an iterative manner where only a single data point is "consumed" each time.

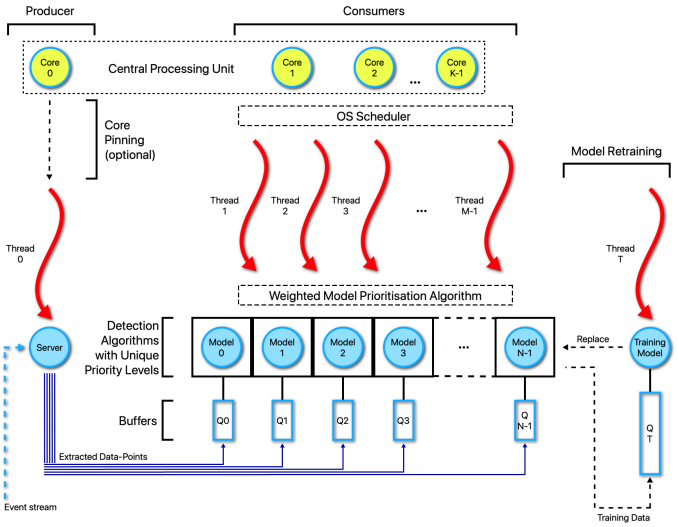

Figure 2. A high-level overview of the scheduler's architecture. The number of consumer threads should be less or equal to the number of detection models.

4.2 Developing a Dynamic User-level Scheduler

The dynamic user-level scheduler is built as an evolution from the baseline system, adding features to increase performance from a security perspective without compromising on it being agnostic to the hardware and (to some extent) the utilised detection algorithm.

Figure 2 displays an overview of the scheduler. The producer thread, represented by *Thread 0* in the figure, has identical responsibilities here as it did in the baseline. Parallel anomaly detection is achieved by multi-threading the consumer side and placing the detection models in a task pool. A detection model can only be accessed by one consumer thread at a time to prevent race conditions.

Notably, the scheduler is designed to be adaptive against the available hardware resources (such as CPU cores and memory) by striving to equally distribute the workload among all consumer threads, ensuring scalability.

4.2.1 Dynamic Resource Prioritisation. Prioritising certain detection models over others increases the overall effectiveness of our IDS architecture from a security perspective. This results in consumer threads executing models of higher priority more frequently than others.

Prioritisation is based on three conditions: (1) Buffer size exceeds 80% of its maximum capacity, hence approaching an overflow situation, (2) Anomaly scores from a detection model surpass a predetermined "soft" threshold value, indicating that an attack might occur, (3) A model's anomaly scores surpass a "hard" threshold value, alarming that the node is under attack.

4.2.2 Minimising Starvation and Overhead. When a consumer thread is ready to perform anomaly detection, it

first calls on the "Weighted Model Prioritisation Algorithm" (WMPA) (see Figure 2), which returns the index for which detection model to be executed next.

An "availability"-flag has been implemented for each detection model indicating the models available for execution. The models' priority level and their index, from Figure 2, are stored as pairs in a separate array. This array is fed to the WMPA, which calculates the cumulative sum of the detection models' priorities and stores them in another array named "sum". WMPA then calls for a random number r, between zero and the sum of all priorities. Lastly, it returns the model at index r from *sum* for execution.

WMPA's most significant trait is that it favours higher-priority models. This may cause low-priority models to be neglected, however, we stand by this decision since it indicates that the model neither produces suspicious anomaly scores nor receives much data.

4.2.3 Model Retraining. Data-driven ML models must be retrained once the initial training set is no longer up-to-date with the incoming data. Otherwise, normal behaviour may be classified as malicious or the opposite. However, retraining is computationally intensive, even when using a lightweight algorithm. Additionally, the solution may not obstruct the IDS from performing intrusion detection.

The retraining phase begins with collecting up-to-date data. Whenever a consumer thread fetches data from the buffer belonging to the model to be retrained, it will also insert a copy of that data into a separate buffer belonging to what is denoted as the *Training Model* in Figure 2.

The training model is stored in a separate data structure to minimise interference between training and other IDS responsibilities. Once enough data has been collected, a new thread spawns to perform the actual training tasks ("Thread T" in Figure 2). Once training is concluded, the outdated detection model may finally be replaced.

5 EXPERIMENTAL METHODOLOGY

5.1 Architectural adaptations

To avoid network inconsistencies becoming a source of error, the producer thread will read incoming communication traffic directly from an IP-traffic file during performance tests. The IP file consists of roughly 190,000 Ethernet packets. As was described in Section 1, our system is agnostic to the type of sensor data, and its specific content is not of particular interest to this work.

Additionally, when retraining is enabled, the retraining task will run continuously in parallel with anomaly detection during the whole test run. One of the four available CPU cores will be dedicated to performing retraining.

The following architectural adaptations were made for the tests: (1) the producer thread dispatches a burst of data points each second uniformly to all buffers, (2) the buffer sizes are fixed at 10,000 elements since our only concern is that the size is not increasing indefinitely, (3) to avoid time slicing, the scheduler will be run using four threads, equal to the number of cores.

5.2 Chosen Detection Algorithm

Our scheduler is designed for compatibility with the class of ML algorithms specified in Section 1. For this work, we have chosen to utilise "PASAD" [3] during testing but any other lightweight data-driven anomaly detection algorithm based on ML could also have been used. As mentioned in Section 1 and 3 we intend to view the detection algorithm as a black box. We wish to emphasise again that analysing the detection capabilities of the ML algorithm (e.g. precision, recall etc.) is not this work's aim, but rather to observe the performance impact of the dynamic user-level scheduler on the IDS and if it can enable the IoT unit to execute ML-based tasks (inference and model retraining).

5.3 Hardware Devices

We tested our solutions on two separate hardware devices, the "Raspberry Pi 4 Model B" and the "NVIDIA Jetson Nano". Both are equipped with a quad-core CPU and 4GB of LPDDR4 RAM while still possessing exciting differences. The Raspberry is equipped with single-channel RAM with a bandwidth of approximately 4.4 GB/s, while the NVIDIA unit has dual-channel RAM with a bandwidth of 25.6 GB/s. For a more extensive specification description, see [13], [11].

We decided to work with high-end IoT devices equipped with a proper operating system and kernel. In other words, our IoT units are more similar to a network router than to, e.g., a temperature sensor. It is more likely that the router manages the IDS since it has access to the network traffic.

6 EXPERIMENTAL EVALUATION

Below, we will go over the results of three experiments; "Maximum throughput measurement", "Evaluation of model prioritization" and "Scale-up".

6.1 Maximum Throughput Measurements

This measurement refers to the maximum number of anomaly scores calculated per second for a specific number of PASAD instances. More precisely, we increase the rate of incoming communication traffic until the consumer(s) starts to miss data points intended for anomaly detection.

Simultaneously for each test run, we increase the number of PASAD instances until the memory is filled. Observe that in the graphs, "anomaly scores" have been denoted as the more general term "departure scores". Furthermore, the producer will loop through the IP traffic file 50 times during each test. To eliminate errors caused by background processes, we repeat the test several times while slowly iterating the input rate until we find the system's maximum throughput.

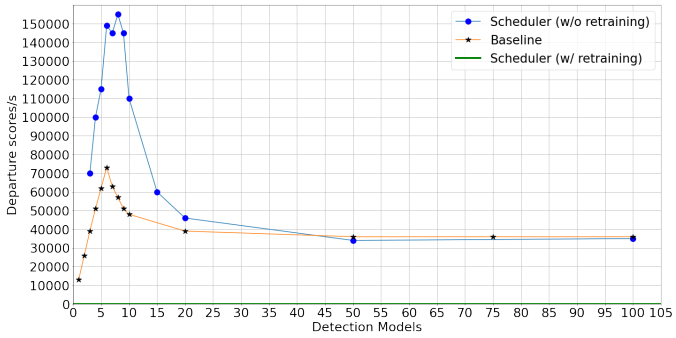

Figure 3. Comparison between the baseline system and scheduler on the Raspberry Pi 4. As this unit proved unable to perform retraining, that line remains at zero for all cases.

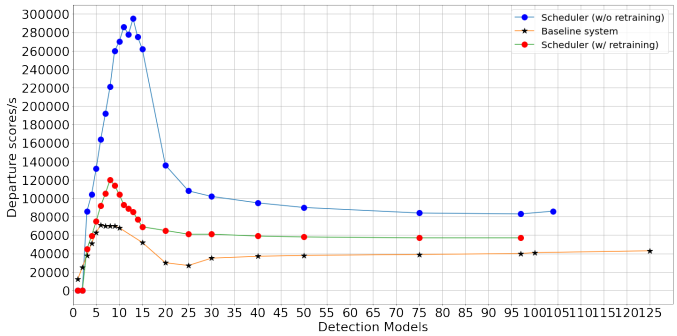

Figure 4. Comparison between the baseline system and scheduler, running on the NVIDIA Jetson Nano.

6.1.1 Raspberry Pi 4.

Figure 3 shows the maximum throughput for a set of PASAD instances on the scheduler and baseline system before the consumer(s) start to miss data.

As observed, when going beyond nine PASAD models, the scheduler hits what appears to be a "performance wall", and eventually behaves identically as the baseline. Also, note that performing retraining in parallel with anomaly detection failed on the Raspberry, as indicated by the green line in Figure 3.

6.1.2 NVIDIA Jetson Nano.

Contrary to the Raspberry, the NVIDIA unit was capable of executing model retraining in parallel with anomaly detection (see Figure 4). Here, the scheduler outperforms the baseline in almost all instances, even with retraining enabled.

6.2 Evaluation of Model Prioritisation

As explained in Section 4.2.1, the scheduler can dynamically allocate resources to nodes needing them. We devised an experiment to quantify this feature's effect.

The conducted methodology is to intentionally cause just one detection model to breach both the soft and hard thresholds by setting these limits to be larger than the scores produced from a model fed with data from our IP-traffic file. A different data stream causing prioritisation is then dispatched to one unique model.

This model immediately breaches the soft threshold upon run-time initialisation and later the hard threshold after looping the traffic file 30 times. The time until alarm detection is measured.

In Table 1, the alarm detection time is listed for each system. Each processed data at its maximum capacity for 75 models, which for both versions of the scheduler is around 84,000 anomaly scores per second. However, the baseline system only manages 39,000 scores per second, affecting the detection speed significantly.

In Table 2, all systems equally produced 39,000 anomaly scores per second instead to eliminate throughput as an error source for the baseline. We see that running the scheduler at a much lower throughput than what it can handle causes the benefits from model prioritisation to almost be eliminated.

Figure 5 displays the maximum buffer sizes during run-time for each model after having conducted the experiment from Table 1 on the scheduler with active prioritisation. The metric for the unique model which intentionally breached both the soft and hard thresholds has been highlighted using a red circle.

The higher prioritisation significantly affected the buffer sizes since consumer threads execute the highly prioritised model more frequently than the others.

Table 1. Time measurements averaged over three runs.

System type	Time until alarm	Delay
Scheduler	98.167 s	-
Scheduler (w/o prioritisation)	104.963 s	≈ 6.8 s
Baseline (w/o prioritisation)	187.252 s	≈ 89.1 s

Table 2. Time measurements averaged over three runs.

System type	Time until alarm	Delay
Scheduler	180.643 s	-
Scheduler (w/o prioritisation)	180.742 s	≈ 0.99 s
Baseline (w/o prioritisation)	187.252 s	≈ 6.61 s

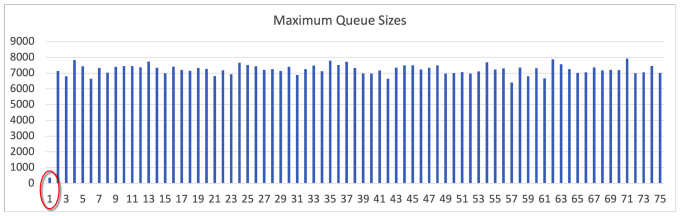

Figure 5. The columns display the largest buffer size for each detection model throughout the complete run-time.

6.3 Scale-up Measurements

Scale-up observes the difference in problem size while execution time remains identical. In our case, scale-up indicates how many more data points the scheduler can perform anomaly detection on compared to the baseline during the same time. Here, 75 PASAD models were declared, and the systems performed anomaly detection by iterating through the IP traffic file for 290 seconds. The test was only performed on NVIDIA Jetson Nano, as the scheduler behaved identically to the baseline on the Raspberry Pi 4.

Results showed that the baseline could analyse ≈10,000,000 data points while the scheduler analysed ≈16,400,000 and ≈12,400,000 with retraining disabled and enabled, respectively. Meaning a scale-up factor of ≈1.64 and ≈1.24 was achieved.

7 DISCUSSION

7.1 Raspberry Pi's Performance Issues

In Figure 3 we see that when going beyond a certain amount of detection models, the scheduler displays zero performance improvement over the baseline. This "performance wall" is most likely a memory-bound issue between the CPU and Random Access Memory (RAM). The graph's behaviour suggests that from ten models onward, we exceed what fits inside the L2 cache and begin accessing the RAM. After roughly 20 models, the RAM is accessed constantly, making memory bandwidth the limiting factor. Additionally, the single memory channel between the L2 cache and RAM causes significant idleness since only one consumer thread may use it at the time.

These issues are not caused by the scheduler itself but rather by the increasing amount of data our detection algorithm (PASAD) instances require. We also believe that the single memory channel is why the Raspberry Pi could not simultaneously perform model retraining and inference. The training phase requires a lot of data, rendering the memory channel fully occupied while buffers are overfilled from continuing insertions by the producer thread.

7.2 The Dynamic User-level Scheduler

The Jetson Nano proved superior to the Raspberry Pi in running the dynamic user-level scheduler despite having slightly less capable CPU cores, a feat most likely due to their differing memory structures.

Firstly, since the Jetson Nano has an L2 cache size of 2MB (double the Raspberry Pi's capacity), it can fit a higher number of PASAD instances inside its cache memory. This is why its throughput decline takes place later, and the higher memory bandwidth also aids performance since the constantly required data can be accessed faster.

Most importantly, however, the Jetson Nano is equipped with dual-channel memory that can provide RAM access for two threads. As Figure 4 shows, this leads to performance never dropping to the same throughput as the baseline system, contrary to the Raspberry Pi. It also enables simultaneous retraining and inference, as one memory channel may be occupied for each of these data-heavy tasks.

Regarding scale-up from Section 6.3, both scheduler versions could process 64% and 24% more anomaly detection, respectively, during the same execution time compared to the baseline. Some of the error sources causing our imperfect results are:

- Overhead from added features, primarily the WMPA, which accounts for all models' availability and priority before each inference calculation.
- Actual implementation of PASAD, which we have no insight into due to efforts of keeping the solution as algorithm agnostic as possible.

7.3 Attack Detection

In Table 1, model prioritisation enabled the alarm to be detected almost seven seconds faster, even though all other parameters remained identical. Figure 5 visualises the reason: the consumer threads have performed more work on the single model set to breach our thresholds.

Table 2 shows that the scheduler can detect the alarm over six seconds faster than the baseline, but interestingly the benefit of model prioritisation disappeared. This is because the scheduler ran at a much lower input rate than it can handle, so the data got processed immediately anyway.

8 CONCLUSION

We have developed our own dynamic user-level scheduler to improve ML-based IDS for IoT which uses PASAD as its detection algorithm. The developed scheduler is agnostic to the hardware and to data-driven and lightweight ML models. The scheduler improves IDS for IoT by enabling on-device retraining parallel to intrusion detection. Our scheduler running on an NVIDIA Jetson Nano achieves an ≈1.24 times performance increase compared to a baseline system while simultaneously retraining ML models and ≈1.64 times performance without retraining. As such, we show that schedulers accounting for the needs of IDS on IoT can speed up the analysis and keep all computations locally, thereby eliminating the need of cloud offloading.

Acknowledgments

The research leading to these results has been partially supported by the Swedish Civil Contingencies Agency (MSB) through the projects RICS2, as well as the CELTIC-NEXT AI-NET-PROTECT (C2019/3-4) project and Clavister.

References

[1] Magnus Almgren, Wissam Aoudi, Robert Gustafsson, Robin Krahl, and Andreas Lindhé. 2018. The Nuts and Bolts of Deploying Process-Level IDS in Industrial Control Systems. In *Proceedings of the 4th Annual Industrial Control System Security Workshop* (San Juan, PR, USA) *(ICSS '18)*. Association for Computing Machinery, New York, NY, USA, 17–24. https://doi.org/10.1145/3295453.3295456

[2] Manos Antonakakis, Tim April, Michael Bailey, Matthew Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. 2017. Understanding the Mirai Botnet. In *Proceedings of the 26th USENIX Conference on Security Symposium* (Vancouver, BC, Canada) *(SEC'17)*. USENIX Association, USA, 1093–1110.

[3] Wissam Aoudi, Mikel Iturbe, and Magnus Almgren. 2018. Truth Will Out: Departure-Based Process-Level Detection of Stealthy Attacks on Control Systems. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (Toronto, Canada) *(CCS '18)*. Association for Computing Machinery, New York, NY, USA, 817–831. https://doi.org/10.1145/3243734.3243781

[4] Chafika Benzaïd and Tarik Taleb. 2020. AI for Beyond 5G Networks: A Cyber-Security Defense or Offense Enabler? *IEEE Network* 34, 6 (2020), 140–147. https://doi.org/10.1109/MNET.011.2000088

[5] Cloudflare. 2022. What is the Mirai Botnet? https://www.cloudflare.com/learning/ddos/glossary/mirai-botnet/. Online; last accessed 3 February 2023.

[6] Hervé Debar, Marc Dacier, and Andreas Wespi. 2000. A revised taxonomy for intrusion-detection systems. In *Annales Des Télécommunications* (Paris, France) *(55)*. Springer-Verlag, New York, NY, USA, 361–378. https://doi.org/10.1007/BF02994844

[7] Sundas Iftikhar, Mirza Mohammad Mufleh Ahmad, Shreshth Tuli, Deepraj Chowdhury, Minxian Xu, Sukhpal Singh Gill, and Steve Uhlig. 2023. HunterPlus: AI based energy-efficient task scheduling for cloud–fog computing environments. *Internet of Things* 21 (2023), 100667. https://doi.org/10.1016/j.iot.2022.100667

[8] Ansam Khraisat and Ammar Alazab. 2021. A critical review of intrusion detection systems in the internet of things: techniques, deployment strategy, validation strategy, attacks, public datasets and challenges. *Cybersecurity* 4, 1 (2021), 1–27. https://doi.org/10.1186/s42400-021-00077-7

[9] McAfee. 2018. Trending: IoT Malware Attacks of 2018. https://www.mcafee.com/blogs/mobile-security/top-trending-iot-malware-attacks-of-2018/. Online; last accessed 3 February 2023.

[10] Henry Muccini and Karthik Vaidhyanathan. 2021. Software Architecture for ML-based Systems: What Exists and What Lies Ahead. In *2021 IEEE/ACM 1st Workshop on AI Engineering - Software Engineering for AI (WAIN)*. IEEE, Madrid, Spain, 121–128. https://doi.org/10.1109/WAIN52551.2021.00026

[11] NVIDIA. 2022. DATASHEET - NVIDIA Jetson Nano. https://developer.nvidia.com/embedded/jetson-nano. Online; last accessed 3 February 2023.

[12] Xuan-Qui Pham and Eui-Nam Huh. 2016. Towards task scheduling in a cloud-fog computing system. In *2016 18th Asia-Pacific Network Operations and Management Symposium (APNOMS)*. IEEE, Kanazawa, Japan, 1–4. https://doi.org/10.1109/APNOMS.2016.7737240

[13] Raspberry Pi. 2019. DATASHEET - Raspberry Pi 4 Model B. https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf. Online; last accessed 3 February 2023.

[14] Thomas Rauber and Gudula Rnger. 2013. *Parallel Programming: For Multicore and Cluster Systems* (2nd ed.). Springer Publishing Company, Incorporated, Berlin, Germany. https://doi.org/10.1007/978-3-642-37801-0

[15] The OWASP IoT Security Team. 2018. OWASP Internet of Things (IoT) Project. https://wiki.owasp.org/index.php/OWASP_Internet_of_Things_Project. Online; last accessed 3 February 2023.

[16] Shreshth Tuli, Sukhpal Singh Gill, Minxian Xu, Peter Garraghan, Rami Bahsoon, Schahram Dustdar, Rizos Sakellariou, Omer Rana, Rajkumar Buyya, Giuliano Casale, and Nicholas R. Jennings. 2022. HUNTER: AI based holistic resource management for sustainable cloud computing. *Journal of Systems and Software* 184 (2022), 111124. https://doi.org/10.1016/j.jss.2021.111124

[17] Liang Xiao, Xiaoyue Wan, Xiaozhen Lu, Yanyong Zhang, and Di Wu. 2018. IoT Security Techniques Based on Machine Learning: How Do IoT Devices Use AI to Enhance Security? *IEEE Signal Processing Magazine* 35, 5 (2018), 41–49. https://doi.org/10.1109/MSP.2018.2825478