Network Intrusion Detection using Machine Learning on Resource-Constrained Edge Devices

Pontus Lidholm[1], Tijana Markovic[2], Miguel Leon[2], Per Erik Strandberg[1]
[1]*Research and Development, Westermo Network Technologies AB,* Västerås, Sweden
[2]*School of Innovation, Design and Engineering, Mälardalen University,* Västerås, Sweden
pontus.lidholm@westermo.com, tijana.markovic@mdu.se, miguel.leonortiz@mdu.se, per.strandberg@westermo.com

***Abstract*—The rapid growth of the Internet has led to the evolution of sophisticated security threats that exploit vulnerabilities within networks. The defence mechanisms must quickly adapt to these new threats to ensure that networks stay secure. One possible mechanism is to use Machine Learning (ML) algorithms to detect malicious activities. The edge devices that control and manage the network, such as routers, already have access to the data that is flowing through the network and may utilize its own computational resources to host ML algorithms and use them to detect intrusions. This paper presents a system for network intrusion detection which is deployed to an edge device and evaluated for live binary classification of network traffic. Different ML algorithms (Decision Tree, Random Forest, and Artificial Neural Network) are evaluated on existing datasets (Westermo and CIC-IDS-2017). Flow-based data pre-processing is performed and different labeling strategies and flow durations are used and compared. The most effective version of each algorithm is implemented and deployed on the Westermo Lynx-3510 routing-capable network switch and system performance is assessed across various scenarios with simulated network attacks. The experiments showed that Random Forest is the best option, closely followed by Decision Tree.**

Keywords: Intrusion Detection, Machine Learning, Edge Computing, Embedded System

I. Introduction

In the current age, both the Internet and companies internal networks are growing in complexity and size. When the Internet was created, it was almost exclusively composed of desktop computers. But nowadays, smaller and less sophisticated devices are being connected. These range from quite advanced devices such as phones, down to Internet of Things (IoT) devices such as lamps and fridges. There is a huge variety of those devices and reliance on them is growing. Companies depend on those devices to function and stay secure. But at the same time, cyberattacks are becoming more complex, while the programs to launch such attacks are getting easier to use. This means that security breaches are more common, which can lead to damages to companies revenue, reputation, and reliance [1].

To protect all the connected devices and to keep up with ever-evolving intrusion methods, one possibility is to arm the network with advanced intrusion detection algorithms. A general trend is to move computing power from the cloud or core networks to the edge to reduce problems of data transportation, as well as latency and privacy problems. Edge network devices, such as routers, can host intrusion detection algorithms since their computational power has increased. Additionally, these devices have already access to the data flowing through the network and may utilize its own computational resources to detect and take action against attacks or intrusions.

In recent years, Machine Learning (ML) has become a popular and effective method for intrusion detection [2], [3], [4]. Various ML algorithms were applied to network datasets to separate normal traffic from malicious traffic, and most commonly used methods include Decision Tree (DT) [5], [6], Random Forest (RF) [7], [8], [9], Artificial Neural Network (ANN) [10], [11], [12], Support Vector Machine (SVM) [13], [14], or K-means [15], [3]. Additionally, the possibility to develop light ML algorithms that can run in a resource-constrained setting is explored in multiple studies. For instance, Roy et al. [16], performed intrusion detection using lightweight methods (K-Nearest-Neighbour (KNN), RF, and Extreme Gradient Boosting) and revealed promising results while being efficient. Doshi et al. [17] researched the performance of different algorithms (KNN, SVM, DT, RF, ANN) which can be applied for low-cost classification of attacks originating from IoT devices.

Most of the applications implemented ML algorithms in the cloud, but bringing them to the edge reduces the need for data transportation and enhances privacy and security of network data. Studies presented in [18], [19], [20] explored intrusion detection using ML in resource-constrained edge device. This paper continues in this tradition and explores containerized intrusion detection on the edge devices for live binary classification of network traffic using different ML algorithms. Commonly used algorithms (DT, RF, and ANN) are trained on the server using two existing datasets with different characteristics (Westermo [21] and CIC-IDS-2017 [22]) and deployed to an edge device. Several novel aspects are introduced, including the utilization of a more recent dataset published in 2023, the analysis of network flows instead of individual network packets, and the evaluation of the proposed system using an industrial simulator comprising multiple physical devices. The main contributions can be summarized as follows:

- A system for intrusion detection on the edge devices is presented, including data reading and pre-processing, as well as ML algorithm training and deployment.
- Evaluation of three ML algorithms is performed on two existing datasets using two labeling strategies and

different flow durations.
- The best performing version of each ML algorithm was deployed to an edge device and evaluated for live network traffic classification, where different network attacks were simulated.

The paper is organized as follows. Section II presents the design and implementation of the system. The experimental settings are presented in Section III, while the results of the experiments and discussion of the results are given in Section IV. Lastly, a conclusion and future work are presented in Section V.

II. System Description

Figure 1 shows all components of the system. First, there is the Server part, where the existing data is used to train a model. The model is deployed to the second part of the system, an edge device, where it is used to classify live network traffic. The two parts are explained in detail in the following subsections.

A. Machine Learning Training on the Server

There are several possible strategies for training and deploying ML in edge networks [23]. Sending data for training from the edge to the cloud requires data transport, whereas training on the edge requires computational resources and time. In this paper, pre-existing datasets are used for ML training on the server and the trained models are exported to the edge device. Python programming language and Scikit Learn[1] are used to develop this part of the system. It consists of four subsequent steps which are explained below.

1) Data pre-processing: Pre-existing raw data is pre-processed to be suitable for ML algorithms.

- *Reading packets* - A cross-platform library libpcap[2] is used to read packets from a dataset file.
- *Packets into flows* - Because of the large number of packets flowing through a network, the packets are processed into flows, i.e. groups of similar packets. This method reduces the amount of data, without losing information. Because of the nature of network traffic, often packets transceived in a communication between two parts are similar. This is especially true for the header of the packets while the payload is different. Therefore, the amount of data can be reduced by grouping packets that cohere to the same communication. This is accomplished using the source IP address, destination IP address, destination port, and protocol according to IP Flow Information Export (IPFIX) standard [24]. Multiple packets, within a specific time frame, are grouped into the same flow if these four fields are equal.
- *Feature engineering* - In this phase features are extracted for each flow, based on the IPFIX standard, RFC 7012 [24]. The full list of features is given in Table I. To ensure that the model does not learn to detect attacks in the specific network topology used in the dataset, IP addresses are completely excluded.

[1]https://scikit-learn.org/

[2]https://www.tcpdump.org/

TABLE I: Features calculated for each flow.

Application		
ID	Feature	Description
1.	src port	Source port of the packets
2.	dst port	Destination port of the packets
3.	protocol identifier	protocol field in the IPv4 header
4.	IP class service	Value of the type of service field
Statistical		
ID	Feature	Description
5.	octet delta count	Total number of bytes
6.	packet delta count	Total number of packets
7.	flow duration	Delta time of first and last packet
8.	min IP total length	Smallest IP header + payload packet
9.	max IP total length	Largest IP header + payload packet
10.	IP header length	Length of the IP header
11.	flow byte rate	Amount of bytes per second
12.	flow packet rate	Amount of packets per second
13.	packet len min	Smallest packet in the flow
14.	packet len max	Largest packet in the flow
15.	packet len mean	Mean packet length
16.	packet len std	Standard deviation of the packet size
17.	flow IAT min	Minimum inter-arrival time of packets
18.	flow IAT max	Maximum inter-arrival time of packets
19.	flow IAT mean	Average inter-arrival time of packets
20.	flow IAT std	Standard deviation inter-arrival time
21.	flow IAT total	Total inter-arrival time of packets
22.	mcast packet count	Amount of multicast packets
23.	mcast octet count	Total size of multicast packets
Flags		
ID	Feature	Description
24.	TCP control bits	Flags contained encoded bit fields
25.	IGMP type	Internet Group Management Protocol field
26.	ICMP type code	Internet Control Message Protocol type, code
27.	FIN flag count	N packets with "No more data from sender"
28.	SYN flag count	N packets with "Synchronise sequence"
29.	RST flag count	N packets with "Reset the connection"
30.	PSH flag count	N packets with "Push Function"
31.	ACK flag count	N packets with "Acknowledgement"
32.	URG flag count	N packets with "Urgent"
33.	CWR flag count	N packets with "Congestion reduced"
34.	ECE flag count	N packets with "ECN Echo"
35.	IPv4 options	IPv4 option encoded in bit fields
36.	minimum TTL	Minimum time to live for the packets
37.	maximum TTL	Maximum time to live for the packets
38.	fragment offset	Data starting position for the packets
39.	fragment flags	Fragmentation properties of the packets
40.	ethernet type	MAC client protocol of the payload

2) Labeling: To enable supervised learning, the dataset requires labels that mark each flow as positive or negative. This paper utilizes two different labeling strategies:

- *Network Security Tools (NST)* [25] - In this approach, flows are labeled based on their sender. If the flow contains packets that are sent from the attacker's PC during the time of an attack and towards the target node, then those flows are labeled as an attack. This should ensure that only the packets which are part of the attack are labeled as such. This could decrease the amount of false positives but also strengthen learning about the attack patterns.

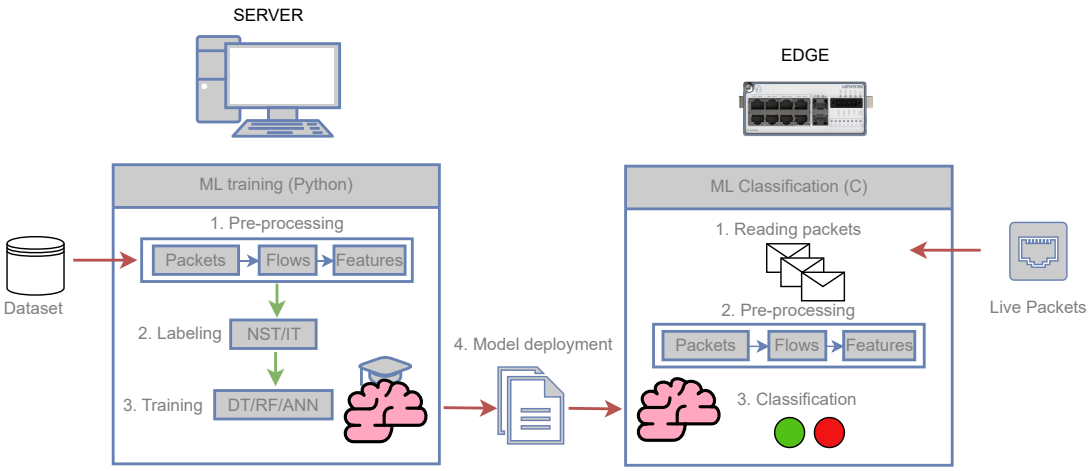


Fig. 1: The architecture of the system.

- *Injection timing (IT)* [26] - A more inclusive strategy, where all flows that include packets that are transmitted during the time of an attack are labeled as an attack. This approach may encapsulate not only attacking patterns, but also the response of the network as this could indicate that an attack is ongoing. This can occur when an attack is indirect, i.e., the attack has no direct target but instead redirects traffic like Man-in-the-Middle (MITM), or when the network has an invalid configuration (e.g., duplicated IP addresses). Therefore this approach tries to include all changes that occur in the network during such attack.

3) Training: ML algorithms are trained using the available data. Different ML algorithms are used in this paper, including:

- *Decision Tree (DT)* [27] is used for classification in a way that it predicts the class label by learning decision rules from the dataset features. This process starts with selecting one feature and separating the instances into groups based on the value of that feature. After that, additional separations are made using different features until the termination criteria are satisfied.
- *Random Forest (RF)* [28] is an ensemble learning algorithm that combines multiple randomized DTs. Each DT uses different features and a separate fraction of the dataset to perform the learning process. The final decision is made by taking the class which is predicted by the majority of independent DTs.
- *Artificial Neural Network (ANN)* [29] is inspired by the human brain. It is composed of artificial neurons which are organized into layers: an input layer, one or more hidden layers, and an output layer. The layers are interconnected and the output of one layer is used as input to the next layer. Those connections are characterized by parameters called weights and the goal of the training process is to find appropriate values for the weights.

A Genetic Algorithm (GA) [30], [31] is used to select the best combination of hyperparameters for each ML algorithm. GA is a population-based algorithm that is formed by a set of individuals, also called population (P), where each individual is a solution to the problem being optimized. In this paper, an individual is formed by the hyperparameters of the ML algorithms, which are randomly initialized under certain constraints (given in Section III-D). After the population is initialized a process, called generation, formed by four steps is repeated several times (number of generations). The steps are:

- *Parent Selection:* Two individuals (P_i and P_j) are selected as parents, using a method called Roulette selection [32].
- *Crossover:* After selecting two individuals in the previous step, the crossover is applied to create new individuals. In this paper, BLX-α [30] is used as the crossover method.
- *Mutation:* Different perturbances are applied to the new individual by modifying the values. Each value has a 5% chance to be modified by using a normal density function with mean equal to $\frac{max-min}{2} + min$ and standard deviation of $\frac{max-min}{6}$ if they are real numbers or rounded if they are integers. For categorical values, each option has equal probability.
- *Replacement:* The mutated individuals replace the worst-performing individuals in the population.

4) Model deployment: The model is exported as a file. The file contains all configurations and data needed to use the model, including the list of required features.

B. Machine Learning on the Edge

The trained ML model is deployed to an edge device, using a container [33], a virtualization technique used to isolate a process from other parts of the operating system. The container is hosted inside the switch and connected to the internal network bridge via a virtual interface as seen in Figure 2. Each physical interface is named using eth, to indicate that it is an Ethernet interface, while the veth, represents a virtual interface. The veth0 is the external interface of the container, while veth1 is the internal, which means that it is the only interface accessible for the model. All packets routed through

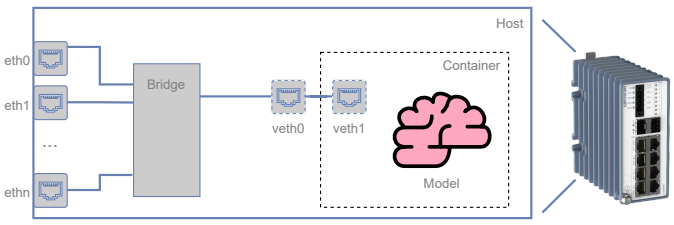


Fig. 2: The container with the model loaded inside. The eth interfaces are physical ports connected to the switch, while the veth0 and veth1 are virtual interfaces existing only in software. The model is isolated from the rest of the software.

the device are also sent into the container, but it is possible to control and filter packets. Also, ports that exist on the device, but that are not connected to the bridge are not accessible to the container, and therefore packets on those interfaces can not be classified.

The classification using the deployed model is performed for each packet entering the container by conducting steps which are listed and explained bellow.

1) Reading packets: The libpcap library is used to read packets directly from a network interface. Because the model is placed inside a container, a software bridge is used to connect the interfaces of the devices to the container. This together with a virtual interface, connects the container to the network from which the packets are collected, as shown in Figure 2.

2) Data pre-processing: The raw data is pre-processed in the same way as on the server (detailed in subsection II-A1).

3) Classification: The trained model is deployed inside the container. Each flow is binary classified and depending on the result an action may be taken.

III. Experimental Settings

In this section, we present the datasets used for the experiments (Section III-A), the data pre-processing steps (Section III-B), the experiments used to evaluate the proposed system (Section III-C), the hyper-parameters (Section III-D), as well as the experimental setup and metrics used for the evaluation (Section III-E).

A. Dataset

The experiments presented in this paper are conducted using two datasets:

- *Westermo network traffic dataset (Westermo)*[3] [21] was recorded over 90 minutes and contains 1.8 million packets with network attacks such as Port Scan, Bad Secure Shell (SSH), and MITM, as well as misconfigurations (network anomalies mostly caused by a human error) such as erroneous or duplicated IP addresses. In addition, there is normal baseline traffic. The dataset was collected using six industrial routers, the factory simulator [34] running on five Raspberry Pi devices, as well as a laptop. The dataset contains three sets of traffic, recorded from different nodes in the network (left, right, and bottom), and all the recordings are used in this paper.
- *Canadian Institute for Cybersecurity Intrusion Detection Evaluation Dataset from 2017 (CIC-IDS-2017)* [22] was recorded over five days and contains 2.8 million network packets with network attacks. It is composed of five subsets, and only Friday subset is used in this paper because it contains Port Scan and Denial of Service (DoS) attacks.

[3]https://github.com/westermo/network-traffic-dataset

TABLE II: Information and distribution of the datasets after pre-processing.

Dataset	Flow Duration	No. of instances	% of attacks using NST	% of attacks using IT
Westermo Left	0.1	112418	24.3%	25.3%
Westermo Left	0.5	50133	22.3%	24.4%
Westermo Left	1	48983	22.9%	25.0%
Westermo Left	2	45811	24.6%	26.4%
Westermo Left	5	41764	28.8%	30.2%
Westermo Left	10	40220	34.2%	35.6%
Westermo Right	0.1	26114	18.2%	22.2%
Westermo Right	0.5	15670	14.3%	20.8%
Westermo Right	1	12774	16.0%	21.7%
Westermo Right	2	11123	16.9%	22.1%
Westermo Right	5	8520	20.9%	25.0%
Westermo Right	10	7574	26.2%	29.5%
Westermo Bottom	0.1	357026	9.2%	19.4%
Westermo Bottom	0.5	152633	9.3%	19.5%
Westermo Bottom	1	93371	10.0%	19.9%
Westermo Bottom	2	55149	11.2%	20.6%
Westermo Bottom	5	29123	14.6%	23.0%
Westermo Bottom	10	19726	20.2%	27.8%
CIC-IDS-2017	0.1	1828246	26.2%	54.1%
CIC-IDS-2017	0.5	1473763	26.1%	56.7%
CIC-IDS-2017	1	1377059	25.1%	56.9%
CIC-IDS-2017	2	1215352	24.8%	54.0%
CIC-IDS-2017	5	1092839	26.0%	53.2%
CIC-IDS-2017	10	987562	27.0%	52.7%

Both datasets are available as raw (PCAP files) and pre-processed data (CSV files). In this paper PCAP files are used.

B. Pre-processing

The following steps are performed during the dataset pre-processing phase:

1) Packets into flows - packets are grouped into flows using different flow durations (0.1, 0.5, 1, 2, 5, and 10 s).
2) Feature extraction - all features presented in Table I are extracted for each flow.
3) Labeling - each flow is labeled using both labeling strategies (NST and IT) using log files about source and timestamps of the attacks.
4) Data normalization - all features are normalized in the range [0, 1] using the Min-Max normalization technique with respect to the training set.

Table II shows total number of instances for each subset using different flow durations, as well as a percentage of attacks in each of them using two labeling strategies. All subsets from the Westermo dataset are unbalanced, regardless of the

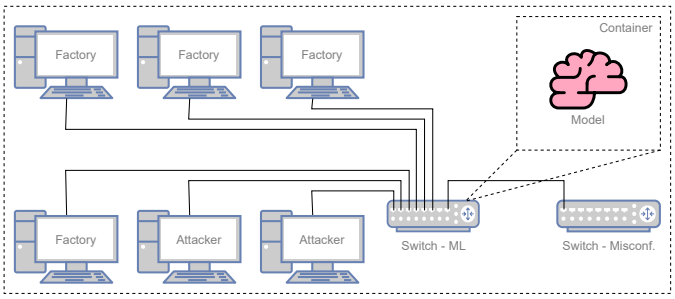


Fig. 3: The topology of the network which was used for the experiment with live network traffic classification.

labeling strategy used, having 9.2% to 35.6% of attack flows. On the other hand, the CIC-IDS-2017 dataset is unbalanced when NST labeling strategy is used (24.8% to 27% of attack flows), but it is well balanced if the IT labeling strategy is applied (53.2% to 56.9% of attack flows).

C. Experiments

To evaluate our approach, we conducted two experiments, one on the existing dataset and one for live network traffic classification.

1) Experiment 1 - Performance on the existing datasets: Three ML algorithms (DT, RF, and ANN) are evaluated on how well they can perform intrusion detection and how different labeling strategies (NST and IT) and different flow durations (0.1, 0.5, 1, 2, 5 and 10) affect their performance. The experiment is carried out on two existing datasets (Westermo and CIC-IDS-2017).

2) Experiment 2 - Performance for live classification: The best performing version of each ML algorithm (DT, RF, and ANN) on the Westermo dataset is evaluated for live network traffic classification on the edge device. To conduct this evaluation, a specific network was set up and its topology can be seen on Figure 3. It was composed of six computers (four were hosting a simulated factory [34] and two were used to launch the attacks) and two network switches. ML algorithms were deployed to the network switched labeled as Switch-ML, while the second network switch (Switch-Misconf.) was used to simulate misconfigurations (misconfigured IP address and duplicated IP address). Three types of attacks were simulated (MITM, DoS, and Port Scan) in three different scenarios. What differs between each scenario is the order the attacks were launched and the time between the start of each attack. The timings used were 33, 60, and 137 seconds. This means that each attack was conducted with the aforementioned amount of seconds in between. The amounts were chosen to decrease the risk of correlation between the scenarios. Each attack is conducted for an equal amount of seconds in each scenario, since it is always initiated using the same command. On the other hand, the duration of misconfigurations is half of the interval before switching the configuration back to normal. Each scenario was repeated three times using the same order and timing. If an attack is detected, the system indicates this by blinking a Light Emitting Diode (LED) on an Ethernet port.

TABLE III: All ML hyperparameters and their constraints that form the individuals in the GA. For ranges, the min and max is listed.

DT		
criterion	options	gini, entropy, log_loss
splitter	options	best, random
max_features	options	sqrt, log2, none
max_depth	range	[10, n_samples]
min_samples_split	range	[2, 40]
min_samples_leaf	range	[1, 20]
max_leaf_nodes	range	[2, n_samples]
min_impurity_decrease	range	[0, 0.01]
min_weight_fraction_leaf	range	[0, 0.01]
RF		
class_weight	options	balanced, subsample, none
max_features	options	sqrt, log2, none
n_estimators	range	[8, 32]
max_depth	range	[10, n_samples]
min_samples_split	range	[2, 40]
min_samples_leaf	range	[1, 20]
max_leaf_nodes	range	[2, n_samples]
max_samples	range	[10, n_samples]
min_impurity_decrease	range	[0, 0.01]
min_weight_fraction_leaf	range	[0, 0.01]
ANN		
activation	options	identity, log, tanh, relu
solver	options	adam, lbfgs
learning_rate	options	constant, invscaling, adaptive
shuffle	options	false, true
hidden_layer_sizes	range	[min, 40]
alpha	range	[0, 0.01]
batch_size	range	[1, n_samples]
learning_rate_init	range	[0.0001, 0.005]
n_layers	range	[1, 5]

D. Genetic Algorithm and Machine Learning Hyperparameters

During training process, the GA generated an initial population of 20, where each gene had a mutation rate of 5%. The genetic operations run over 100 generations. The individual is formed of different ML hyperparameters which can be found in Table III, together with their search space.

E. Experimental description and metrics

Experiment 1 was performed on a laptop with Intel i7-1255U processor and 16 GB DDR4 memory. Python programming language and Scikit Learn[4] ML library were used to implement the experiment. The experiment is performed with 70-20-10% of data for training, validation, and testing respectively.

Experiment 2 was performed on the edge device. The edge device used in this paper is provided, developed, and produced by Westermo Network Technologies. Out of a plethora of devices developed by Westermo, only the Lynx-3510 is used. This routing-capable network switch is powered by NXP i.MX8 Nano, a quad-core processor running at 1.4 GHz, with a total amount of 512 MB of RAM and 128 MB of flash storage. The device operates on the WeOS 5 operating system, which

[4]https://scikit-learn.org/

is based on Linux and optimized for embedded systems. It is capable to host LXC-containers, within which the model is placed.

The performance measure used throughout the entire paper is F1-score, given in the Eq. (1)

$$F1 = \frac{2 \times TP}{2 \times TP + FP + FN} \quad (1)$$

where TP, TN, FP, and FN, stand for True Positive, True Negative, False Positive, and False Negative predictions, respectively.

IV. Results and Discussion

The results of the experiments described in Section III-C are given and discussed in this section. Firstly, the results of the ML algorithms on the existing datasets and how they are affected by different labeling strategies and flow durations are given in Section IV-A. Secondly, the performance of the ML algorithms for live network traffic classification and time/memory usage, is evaluated in Section IV-B.

A. Performance on the existing datasets

The results of the three ML algorithms using the different labeling strategies as well as different flow durations on Westermo dataset are given in Fig. 4, while the results on CIC-IDS-2017 dataset are given in Fig. 5.

The results on the Westermo dataset show that DT and RF are better than ANN in almost all combinations of subset, labeling strategy, and flow duration (30 out of 36 options with DT, and 32 out of 36 options with RF). If RF is compared against DT, it can be noticed that RF is slightly better (24 out of 36 options). If different subsets of the Westermo dataset are considered, it can be seen that performance of all ML algorithms on the bottom subset is lower than on the left and right subsets. The reason is that the bottom subset has the lowest portion of attacks (see Table II). This is confirmed by DF and RF obtaining the best performance on the left subset, which has the highest percentage of attacks. If labeling strategies are considered it can be concluded that all ML algorithm obtain better results using NST labeling strategy. With regard to flow duration, DT and RF manage to maintain a similar performance regardless of the flow duration, except for bottom subset, where considerable improvement is achieved with longer flow durations. On the other hand, ANN performance is improved with longer flows in almost all the cases.

If the results on CIC-IDS-2017 dataset are considered, we can observe that all algorithms obtained similar performances when using IT labeling strategy, which gives a balanced data (see Table II). However, when using NST, ANN obtained the worst performance, while RF obtained the best. In the case of this dataset, flow duration does not have a big impact on the performance of the ML algorithms.

TABLE IV: Time and memory consumption of the ML algorithms when performing live intrusion detection on the edge device. Time is given in microseconds (μs) for the different steps. The recording was conducted on the edge device, using run 1 from the 60-second interval scenario, containing 52901 packets. The flow allocation time is measured for each packet when they were compared, and placed into a flow. The feature extraction and classification times are measured per flow. Peak memory and total memory were measured for the complete process.

Algorithm	Step	Time (μs)		Memory	
		Mean	Std.	Peak	Total
DT	Flow allocation	6.91	4.64		
DT	Feature extraction	6.66	2.13	16.1 MB	25.2 MB
DT	Classification	5.59	8.5		
RF	Flow allocation	7.27	4.51		
RF	Feature extraction	6.21	1.75	17.0 MB	25.7 MB
RF	Classification	13.93	15.66		
ANN	Flow allocation	7.07	4.35		
ANN	Feature extraction	6.42	1.94	16.5 MB	38.0 MB
ANN	Classification	40.94	2.48		

B. Performance for live classification

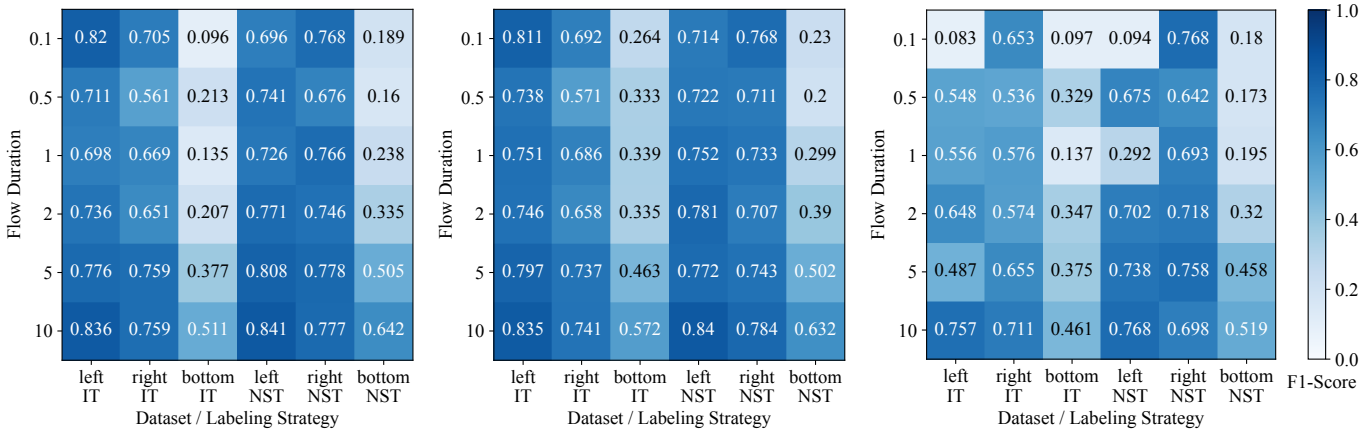
The most effective version of each algorithm was deployed to the edge device. For all three algorithms the best options were the ones trained on the Westermo Left using NST labeling strategy and flow duration of 10. These DT, RF, and ANN had files sizes of 64.4 kB, 1.4 MB, and 51.1 kB respectively. The RF consisted of 26 DTs and the ANN had 5 hidden layers, each comprising 37, 19, 10, 5, and 3 hidden neurons respectively.

Figure 6 shows the results of live network traffic classification during simulation of three different scenarios. The results indicate that Port Scan is successfully detected by all the ML algorithms, in all runs in all scenarios. For DoS attack, DT and RF are always successful, while ANN fails to detect it. DT and RF are sometimes able to detect MITM attack. When it comes to misconfigurations, all the ML algorithms showed poor performance, with DT and RF being able to detect it at least in some cases (DT in Scenario 1 and 3, and RF in Scenario 2). As a final remark, we can see that with the longer interval we have more false positives.

If time consumption is considered (Table IV), we can see that DT is the fastest algorithm, followed by RF, while ANN is the slowest one. On the other hand, the memory consumption (Table IV) is consistent among the algorithms when considering the peak time, while for the total time ANN takes around 50% more memory than DT and RF.

V. Conclusions and Future Work

This paper presents a system for network intrusion detection using machine learning on resource-constrained edge devices. The architecture of the system has two parts, one on a server where the ML model is trained using existing datasets and the second one on the edge device where the ML model is used.



(a) Decision Tree (b) Random Forest (c) Artificial Neural Network

Fig. 4: F1-Score of DT, RF and ANN using Westermo left, right, and bottom subsets.

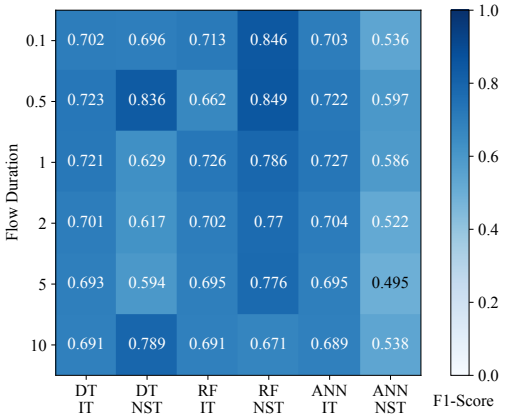


Fig. 5: F1-Score of DT, RF, and ANN using CIC-IDS-2017 dataset.

The system is evaluated in two experiments. First, we evaluate DT, RF, and ANN on the existing datasets using different labeling strategies and flow durations. In particular it can be concluded that RF has the best performance, closely followed by DT. Second, we ran the system on an edge device where live packets were classified and found that DT and RF have similar performance in terms of correctly detected attacks, as well as memory consumption. However, DT is a faster option than RF, while ANN showed the worst performance in all comparisons. It is important to mention that all the ML algorithms fail to detect network anomalies (misconfigurations) and some further research in this direction is needed.

As a future work we plan to investigating how the different features influence the system and if performance can be improved by reducing the number of features used. Additionally, we would like to deploy our framework in a federated learning setup, and explore additional ML algorithms, such as recurrent neural networks.

Acknowledgment

This work has been partially supported by the H2020 ECSEL EU projects Intelligent Secure Trustable Things (InSecTT) and Distributed Artificial Intelligent System (DAIS). InSecTT (www.insectt.eu) has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 876038 and DAIS (https://dais-project.eu/) has received funding from the ECSEL JU under grant agreement No 101007273. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Austria, Sweden, Spain, Italy, France, Portugal, Ireland, Finland, Slovenia, Poland, Netherlands, Turkey. The document reflects only the author's view and the Commission is not responsible for any use that may be made of the information it contains.

Parts of the implementation were developed in a Master's thesis [35].

References

[1] W. Tounsi and H. Rais, "A survey on technical threat intelligence in the age of sophisticated cyber attacks," *Comput. Secur.*, vol. 72, 2018.

[2] S. Wang, J. F. Balarezo, S. Kandeepan, A. Al-Hourani, K. G. Chavez, and B. Rubinstein, "Machine learning in network anomaly detection: A survey," *IEEE Access*, vol. 9, pp. 152 379–152 396, 2021.

[3] M. Leon, T. Markovic, and S. Punnekkat, "Comparative evaluation of machine learning algorithms for network intrusion detection and attack classification," in *2022 international joint conference on neural networks (IJCNN)*. IEEE, 2022, pp. 01–08.

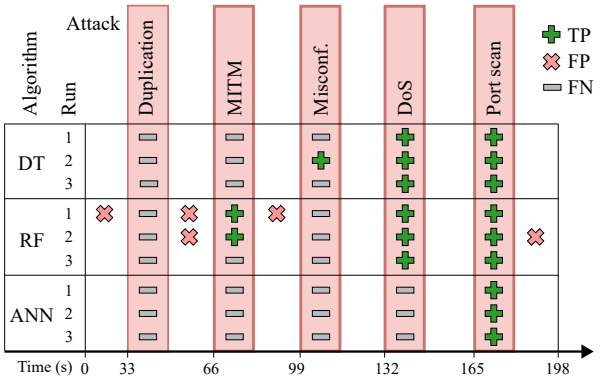
[4] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," *IEEE Communications surveys & tutorials*, vol. 18, no. 2, pp. 1153–1176, 2015.

[5] M. Kumar, M. Hanumanthappa, and T. V. S. Kumar, "Intrusion detection system using decision tree algorithm," in *2012 IEEE 14th International Conference on Communication Technology*, 2012, pp. 629–634.

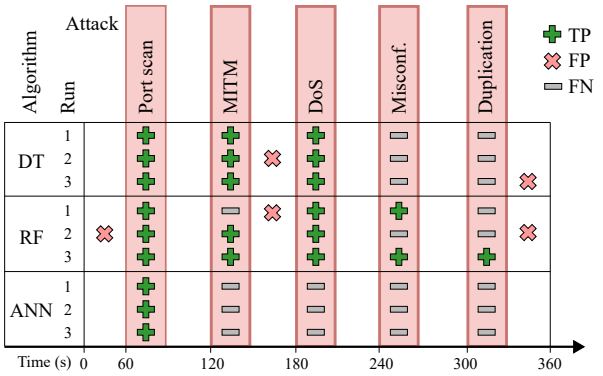
[6] S. Shilpashree, S. Lingareddy, N. Bhat, and G. Kumar, "Decision tree: A machine learning for intrusion detection," *International Journal of Innovative Technology and Exploring Engineering (IJITEE)*, vol. 8, no. 6S4, pp. 1126–1130, 2019.

[7] N. Farnaaz and M. Jabbar, "Random forest modeling for network intrusion detection system," *Procedia Computer Science*, vol. 89, 2016.

[8] T. Markovic, M. Leon, D. Buffoni, and S. Punnekkat, "Random forest based on federated learning for intrusion detection," in *IFIP International Conference on Artificial Intelligence Applications and Innovations*. Springer, 2022, pp. 132–144.



(a) Scenario 1: 33 seconds interval



(b) Scenario 2: 60 seconds interval

(c) Scenario 3: 137 seconds interval

Fig. 6: Results from live network traffic classification on the edge device. A green plus during an attack indicates a true positive, a gray minus means failure to detect attack, and a red x indicates false positives.

[9] A. D. Ghadim, T. Markovic, M. L. Ortiz, D. Söderman, and P. E. Strandberg, "Federated learning for network anomaly detection in a distributed industrial environment," in *International Conference on Machine Learning and Applications 23*, December 2023. [Online]. Available: http://www.es.mdu.se/publications/6853-

[10] G. Poojitha, K. N. Kumar, and P. J. Reddy, "Intrusion detection using artificial neural network," in *2010 Second International conference on Computing, Communication and Networking Technologies*, 2010.

[11] S. Behera, A. Pradhan, and R. Dash, "Deep neural network architecture for anomaly based intrusion detection system," in *2018 5th International Conference on Signal Processing and Integrated Networks (SPIN)*. IEEE, 2018, pp. 270–274.

[12] B. Ingre and A. Yadav, "Performance analysis of NSL-KDD dataset using ANN," in *2015 international conference on signal processing and communication engineering systems*. IEEE, 2015, pp. 92–96.

[13] S. Revathi and A. Malathi, "A detailed analysis on NSL-KDD dataset using various machine learning techniques for intrusion detection," *International Journal of Engineering Research & Technology (IJERT)*, vol. 2, no. 12, pp. 1848–1853, 2013.

[14] T. A. Tuan, H. V. Long, L. H. Son, R. Kumar, I. Priyadarshini, and N. T. K. Son, "Performance evaluation of Botnet DDoS attack detection using machine learning," *Evolutionary Intelligence*, vol. 13, no. 2, pp. 283–294, 2020.

[15] V. Kumar, H. Chauhan, and D. Panwar, "K-means clustering approach to analyze NSL-KDD intrusion detection dataset," *International Journal of Soft Computing and Engineering (IJSCE) ISSN*, pp. 2231–2307, 2013.

[16] S. Roy, J. Li, B.-J. Choi, and Y. Bai, "A lightweight supervised intrusion detection mechanism for IoT networks," *Future Generation Computer Systems*, vol. 127, pp. 276–285, 2022. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0167739X21003733

[17] R. Doshi, N. Apthorpe, and N. Feamster, "Machine learning DDoS detection for consumer internet of things devices," in *2018 IEEE Security and Privacy Workshops (SPW)*, 2018, pp. 29–35.

[18] A. Huč, J. Šalej, and M. Trebar, "Analysis of machine learning algorithms for anomaly detection on edge devices," *Sensors*, vol. 21, no. 14, p. 4946, 2021.

[19] M. Tsukada, M. Kondo, and H. Matsutani, "A neural network-based on-device learning anomaly detector for edge devices," *IEEE Transactions on Computers*, vol. 69, no. 7, pp. 1027–1044, 2020.

[20] M. Eskandari, Z. H. Janjua, M. Vecchio, and F. Antonelli, "Passban ids: An intelligent anomaly-based intrusion detection system for iot edge devices," *IEEE Internet of Things Journal*, vol. 7, no. 8, 2020.

[21] P. E. Strandberg, D. Söderman, A. Dehlaghi-Ghadim, M. Leon, T. Markovic, S. Punnekkat, M. H. Moghadam, and D. Buffoni, "The westermo network traffic data set," *Data in Brief*, vol. 50, p. 109512, 2023, data available online at: https://github.com/westermo/network-traffic-dataset, last accessed 2024-01-10.

[22] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization." *ICISSp*, vol. 1, pp. 108–116, 2018, data available online at: https://www.unb.ca/cic/datasets/ids-2017.html, last accessed on 2023-11-07.

[23] H. Hua, Y. Li, T. Wang, N. Dong, W. Li, and J. Cao, "Edge computing with artificial intelligence: A machine learning perspective," *ACM Computing Surveys*, vol. 55, no. 9, pp. 1–35, 2023.

[24] B. Claise and B. Trammell, "Information Model for IP Flow Information Export (IPFIX)," RFC 7012, 2013. [Online]. Available: https://www.rfc-editor.org/info/rfc7012

[25] J. L. Guerra, C. Catania, and E. Veas, "Datasets are not enough: challenges in labeling network traffic," *Computers & Security*, 2022.

[26] A. Lemay and J. M. Fernandez, "Providing {SCADA} network data sets for intrusion detection research," in *9th Workshop on Cyber Security Experimentation and Test (CSET 16)*, 2016.

[27] J. R. Quinlan, "Decision trees and decision-making," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 20, no. 2, pp. 339–346, 1990.

[28] S. Russell and P. Norvig, "Artificial Intelligence: A Modern Approach," in *Malaysia: Pearson Education Limited.*, 2016.

[29] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[30] S. Picek, D. Jakobovic, and M. Golub, "On the recombination operator in the real-coded genetic algorithms," in *2013 IEEE Congress on Evolutionary Computation*, pp. 3103–3110, ISSN: 1941-0026.

[31] D. Goldberg, "Genetic algorithm in search, optimization and machine learning," in *Addison-Wesley, New York*, 1989.

[32] S. L. Yadav and A. Sohal, "Comparative study of different selection techniques in genetic algorithm," *International Journal of Engineering, Science and Mathematics*, vol. 6, no. 3, pp. 174–180, 2017.

[33] O. Oleghe, "Container placement and migration in edge computing: Concept and scheduling models," *IEEE Access*, vol. 9, pp. 68 028–68 043, 2021.

[34] A. Dehlaghi-Ghadim, A. Balador, M. H. Moghadam, H. Hansson, and M. Conti, "ICSSIM—a framework for building industrial control systems security testbeds," *Computers in Industry*, vol. 148, 2023.

[35] P. Lidholm and G. Ingletto, "Anomaly detection for network traffic in a resource constrained environment," Master's thesis, Mälardalen University, 2023.