Generalizing Supervised Learning for Intrusion Detection in IoT Mesh Networks

Hossein Keipour[1(✉)], Saptarshi Hazra[3], Niclas Finne[3], and Thiemo Voigt[2,3(✉)]

[1] Blekinge Institute of Technology, 37179 Karlskrona, Sweden
`hoke19@student.bth.se`
[2] Uppsala University, 75105 Uppsala, Sweden
[3] Research Institutes of Sweden (RISE), 16440 Stockholm, Sweden
{`saptarshi.hazra, niclas.finne, thiemo.voigt`}`@ri.se`

Abstract. IoT mesh networks typically consist of resource-constrained devices that communicate wirelessly. Since such networks are exposed to numerous attacks, designing intrusion detection systems is an important task and has attracted a lot of attention from the research community. Most existing work, however, has only considered a few network topologies, often also assuming a fixed number of nodes. In this paper, we generate a new large attack dataset, using Multi-Trace, a tool that we recently devised to generate traces to train machine learning algorithms. We show that using more and more diverse training data, the resulting intrusion detection models generalize better compared to those trained with less and less diverse training data. They even generalize well for larger topologies with more IoT devices. We also show that when we train different machine learning methods on our dataset, the resulting intrusion detection systems achieve very high performance.

Keywords: Internet of Things ; 6LoWPAN ; RPL ; Intrusion Detection System ; Blackhole ; Machine learning ; Deep Learning ; Dataset.

1 Introduction

The IoT [4] internetworks everyday objects equipped with sensing, computation, and communication capabilities. The number of connected IoT devices is expected to increase to 38.6 billion by 2025 and an estimated 50 billion by 2030 [12]. IoT is playing a vital role with applications in a variety of areas, such as health care, industrial systems, smart homes, transportation networks, energy management, smart cities, and the energy industry [15].

Recently, we have been witnessing a tremendous increase in attacks on Internet infrastructures. For instance, IoT devices were hacked and used in a DDoS attack by the Mirai botnet in October 2016. This example shows that IoT security is a major concern since it does not only affect the IoT networks themselves but can have a strong negative impact on the Internet.

To prevent such attacks, IoT networks typically employ three main components [6]: First, there are components that aim at *preventing* attacks. Second, there are components that are tasked to *detect* attacks and finally, there are components that should *mitigate* attacks. Once the first line of defense, i.e., the attack prevention component, has been crossed, the second line of defense's task is to detect any suspicious behaviour. The latter is done via intrusion detection systems (IDS). Due to the resource constraints of low-power embedded platforms, designing efficient intrusion detection systems is extremely challenging [6].

Recent research has designed a number of IDS for IoT networks. Most of them, however, are trained on small datasets or datasets not specifically collected for resource-constrained IoT networks [3, 10, 23, 25]. Others do not consider the computational constraints of low-power IoT networks [13, 22].

In this study, we design an ML-based IDS for 6LoWPAN. To train our IDS models, we implement various IoT network scenarios with and without attacks in the Cooja simulator [17] and use our Multi-Trace extensions to generate the dataset [11]. We depart from previous work in that we generate and use a new dataset that covers many different IoT mesh network topologies.

We use this new dataset to train a variety of ML algorithms. Using DNN and SVM-RBF learning algorithms, we achieve a high accuracy (97%) and precision of (95%) and (96%) respectively. RFC also achieves 98% accuracy and 96% precision. RFC models have less complexity and are able to run on resource-constrained low-power IoT mesh networks. Meanwhile, DNN and SVM-RBF need to be implemented in collaboration with the cloud. We train the machine learning algorithms using our dataset. Our results show that with more and more diverse scenarios, i.e., different topologies with different attackers, the machine learning algorithms generalize better, i.e., they achieve better results on the test data.

We also expect our dataset and dataset generation approach to be useful for researchers in the field of security for low-power wireless multi-hop networks and help them to evaluate their machine learning models on this larger dataset.

Contributions. We make the following contributions in this paper:

- We generate a new attack dataset for IoT multi-hop networks that is larger and has more distinct network topologies than previous datasets.
- We show that using more and more diverse training data, the resulting models generalize better compared to those trained with less and less diverse training data. Our models even generalize well when the test set consists of larger topologies with more nodes than the training set.
- We also show that when we train different machine learning methods on our dataset, the resulting intrusion detection systems achieve very high performance.

Outline. After presenting some background, we outline our methodology in Section 3. Section 4 discusses how we generate our attack traces and the following section presents the IDS design. Section 6 is concerned with the ML algorithms. While Section 7 presents our experimental results, Section 8 concludes the paper.

2 Background

2.1 IoT Mesh Networks and RPL Routing

Low-power and lossy networks (LLNs) are a type of IoT network whose nodes collaboratively interact and perform various tasks autonomously. As they are often resource-constrained and battery-powered it is important to keep the energy consumption, in particular for communication, low. Due to the limited IPv4 address space, IoT networks typically use IPv6 to connect directly to the Internet using open standards.

In multi-hop wireless networks, not all nodes are able to reach the gateway with their wireless transmissions. Hence, nodes need to forward packets on behalf of other nodes towards the gateway. The latter is often called sink or root node.

RPL is a widely used routing protocol for LLNs. Usually, one root node is connected to the Internet via an IPv6 Border Router (6BR). Nodes in RPL use a Destination Oriented Directed Acyclic Graph (DODAG) to create and maintain the network topology. The distance from the root is calculated by a scalar value called rank that determines nodes' position in the network. A node's rank value increases with the distance from the root.

2.2 Attacks on RPL

LLNs face two main security challenges. The first challenge is node removal or attacks on nodes' hardware [20]. Second, there are a large number of attacks at the network layer that include Denial of Service (DoS) attacks, sinkhole, blackhole, hello flooding, clone ID, local repair, sybil, and selective forwarding attacks [21, 28]. We categorize attacks on RPL into three classes: (i) attacks that affect traffic, (ii) attacks that disrupt the network topology, and (iii) attacks that aim to affect network resources [2]. Figure 1 shows attacks on RPL and the related categories [21].

2.3 Blackhole Attack

In a blackhole attack the malicious node drops all packets it is supposed to forward to the root node. Malicious nodes accomplish this in two steps. First, the attacker node advertises a fake low-rank value to attract neighbors to select it as their parent (sinkhole attack). Second, it could drop some of the packets based on predefined rules (selective forwarding attack), or drop all packets originated from other nodes (blackhole attack, see Algorithm 1). Blackhole attacks affect network performance. Sharma et al. demonstrate a 26% performance reduction caused by a blackhole attack [24]. Pasikhani et al. [18] compare different attacks on RPL and evaluate their impact. They show that blackhole attacks have a high impact on packet delivery rate and delay and medium impact on control packet overhead and battery drain.

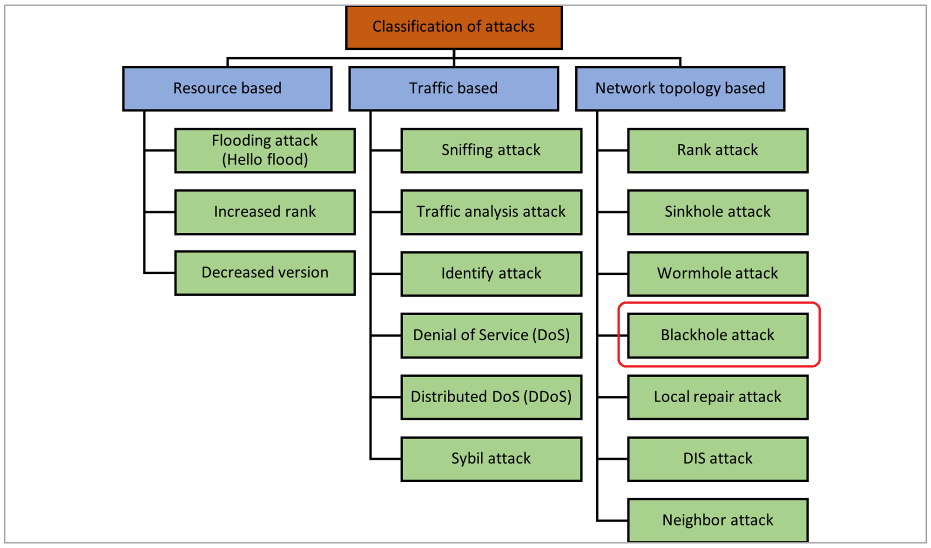


Fig. 1. RPL Attacks classification

Algorithm 1 Attack Scenario: Blackhole Attack

```
Require: Attacker Node-ID
  if Node-ID Matches Attacker Node-id then
      Decrease the rank value
      Keep parents with higher rank
      Drop all packets originated from other nodes except parent
  else
      Keep defined rank calculations
  end if
```

3 Methodology

In this study, we use a simulator to design, run and collect network data from different scenarios. We have earlier devised Multi-Trace [11] an extension to Cooja that collects simulation logs at multiple levels at the same time. We use Contiki's network simulator Cooja to run our experiments and Multi-Trace to generate our dataset. Figure 2 shows the steps from the generation of the dataset to training and evaluating the ML models.

We first create initial simulations with different number of nodes. Then using Multi-trace, we create simulations with random positions of the nodes and a random attacker node in each simulation. Then we run all simulations using Cooja's non-GUI mode for fast performance. In the next step, we collect the raw data from the simulations' output. We then normalize the raw data and extract the features. In the data labeling step, we label each node's traces as attack or non-attack. Using the training dataset we train our ML models and then evaluate the ML models' performance using the test dataset.

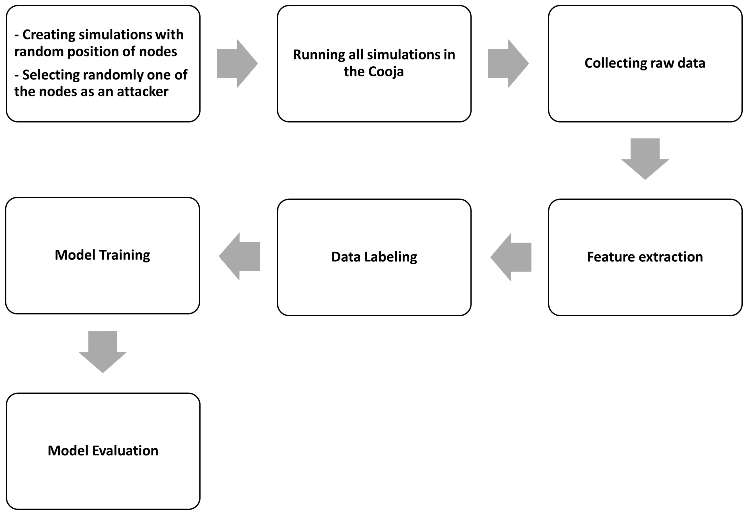

- Creating simulations with random position of nodes
- Selecting randomly one of the nodes as an attacker
Running all simulations in the Cooja
Collecting raw data
Feature extraction
Data Labeling
Model Training
Model Evaluation

Fig. 2. Approach to design our intrusion detection system. The major novelty is that we use simulations to create a large dataset with varying number of nodes.

4 Attack Trace Generation

4.1 Simulation Setup Tool

We use Contiki's network simulator Cooja [17] to simulate embedded devices running the Contiki OS [9]. Using Cooja's Multi-Trace extension [11], we create our own IDS dataset. With this dataset, we test and evaluate our proposed ML models.

The advantage of our dataset compared to previous efforts is that it contains much more variability in terms of the number of nodes and their positions. To achieve that, we simulate several multi-hop IoT networks with a different number of nodes, random node positions, and we select the attacker node randomly.

4.2 Data Collection

To collect network data, we conduct simulations using the Cooja simulator. We call each network topology a scenario. Each scenario has several nodes positioned randomly and one attacker node selected randomly from the nodes as shown in Figure 3.

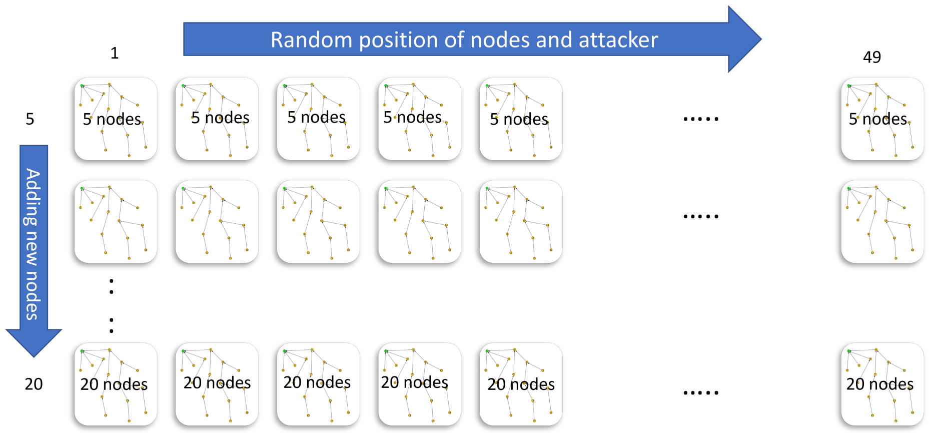


Fig. 3. Different scenarios used for data collection

We collect simulations data from 735 attack scenarios (blackhole attack) and 661 non-attack scenarios (overall 1396 scenarios). Table 1 shows the Cooja simulation setup for the experiments.

Table 1. Cooja simulation setup

Parameters	**Values**
Node Operating System	ContikiNG
Routing Protocol	RPL
Radio Medium Model	Unit Disk Graph Medium (UDGM) Distance Loss
Communication Range	50 m, Interference: 100 m
Number of nodes	5 to 20 (root is excluded)
Distance Between Neighbors	Random <50 m
Number of root nodes	1
Number of Attacker Nodes	1
MAC Driver	CSMA/CA
Network Layer	ContikiRPL
Network Driver	Sicslowpan
Transport Layer	UDP
Traffic model	Constant Bit Rate
Attack start time	30 min after start of simulation
Simulations Duration	60 min

Placing Nodes Randomly. To place the nodes in random positions in each scenario, we use a Python script that receives an initial scenario as input and then creates a specific number of new topologies with random nodes' placement. We put some limitations on the nodes' positions to keep the nodes in the radio range of at least one node to avoid that the network partitions.

Running Simulations in Cooja. We use a Python script to run all simulations automatically. This script uses Cooja's non-GUI ability to run simulations faster and automatized. The results are a collection of simulations statistics stored in separate folders. Data collection starts after the network is stable. We define a network as stable when the root node has received at least one message (data packet) from each node.

Parsing Simulations' Statistics. After running all simulations, we parse the logs. The logs for each scenario contain the run-time, pcap file, and mote output. We modified Contiki's RPL code to report DODAG statistics. Table 2 shows the parsed primary features from the nodes' RPL statistics.

Table 2. Parsed primary features from nodes' RPL statistics. The statistics are reported by each node once per minute.

Time	Time stamp of events in the simulation
Mote	Nodes' ID (number)
Seq	Sequence number of the statistics report from the node
Rank	Rank value of the node
Version	DODAG version
DIS-R	Number of DIS messages received last minute
DIS-S	Number of DIS messages sent last minute
DIO-R	Number of DIO messages received last minute
DIO-S	Number of DIO messages sent last minute
DAO-R	Number of DAO messages received last minute

5 Design of the Intrusion Detection System

5.1 Data Pre-processing

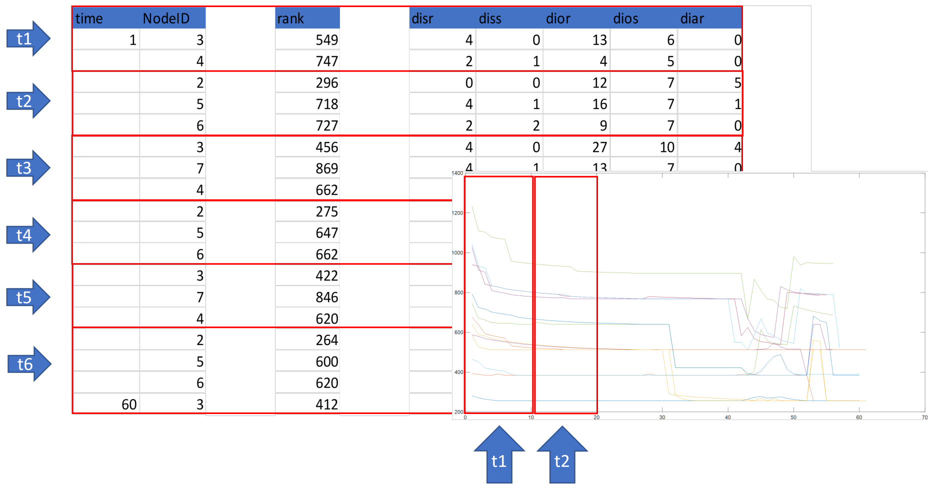
After collecting the data, we normalize it. Using normalization makes the ML model training faster and achieves better generalization accuracy [27]. In this study we use "scaling to a range" normalization method to map our data for each feature to the range of [-1, +1] using Equation 1.

$$x^{'} = 2 * \frac{x - x_{min}}{x_{max} + 1} - 1 \tag{1}$$

5.2 Chunking

After normalizing the collected data, we split our data traces into intervals (chunks) of equal duration to analyse the selected features' changes between these intervals. We select three different chunk sizes that lead to different number of chunks (three, six, and ten). For example, to get ten chunks, we divide the data traces into ten time intervals, each with a duration of six minutes. The attack will starts after 30 minutes of simulation time.

To ensure that the network is stable we do not consider data trace values for following the intervals: the first interval when have three chunks, the first two intervals when we have six chunks, and the first three intervals when we have 10 chunks. Then we average the features' values in the remaining time intervals and store them in a vector. We form a feature matrix that consists of the vectors of each node in each scenario. Figure 4 shows the process of using chunks and removing the initial ones (t1 and t2 when using six chunks).



time	NodeID	rank	disr	diss	dior	dios	diar
1	3	549	4	0	13	6	0
	4	747	2	1	4	5	0
	2	296	0	0	12	7	5
	5	718	4	1	16	7	1
	6	727	2	2	9	7	0
	3	456	4	0	27	10	4
	7	869	4	1	13	7	0
	4	662					
	2	275					
	5	647					
	6	662					
	3	422					
	7	846					
	4	620					
	2	264					
	5	600					
	6	620					
60	3	412					

Fig. 4. Process of using chunks (e.g., six chunks)

5.3 Feature Engineering

Feature engineering categorizes the collected data to improve classification performance. Towards this end, we implement the two steps of feature extraction and feature selection. We perform feature extraction using the chunking method described above.

Then we use Principal Component Analysis (PCA) to reduce the feature dimension. The dimension reduction compresses the data and reduces the required storage space. It also helps to decrease the computational complexity of the ML model and running the ML model faster.

5.4 Data Labelling

After having cleaned the data, we label it. We label the nodes affected by the attacker as under attack nodes and the nodes that are not affected by the attacker in attack scenarios and normal scenarios as non-attack nodes.

We use the variance in the node's rank between pre-defined intervals (chunks) as a metric to understand whether a node is affected by an attack or not.

The blackhole attack first uses the Rank attack to attract neighbor nodes to select the intruder node as a parent. So the vulnerable nodes' rank decreases after the attack has started. Figure 5 shows how the blackhole attack changes the rank value of the intruder's neighbors. It is also visible that after starting the attack (30 minutes in the simulation), some nodes are not affected by the attack and hence their rank value does not change (nodes presented by dashed line) and some nodes experience a dramatic change in their rank value (nodes presented by solid line). When a node is affected by the attack, its children will also be affected.

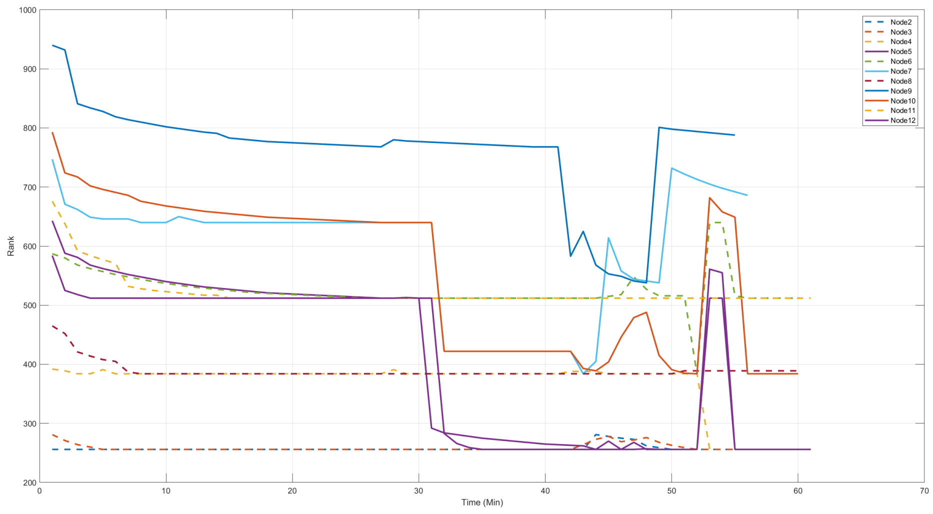


Fig. 5. Rank changes after the starting of an attack

6 Machine Learning Models

After preparing the feature matrix, we use ML methods to perform the classification task. We use this feature matrix to train our predictive model. The model then maps the input data to one of the attack or non-attack classes.

There are several ML classifiers used by researchers to train the ML models for IDS: According to [8, 14, 16, 26] both supervised (e.g., SVM, DNN, RFC) and unsupervised (e.g., K-means, K-Dimensional Tree.) ML algorithms can be used for IDSs.

In this study, we use DNN, RFC, and SVM to detect an attack in the network because of their success in binary classification (two-class) problems.

We use the Keras library to create the ML models in Python. Keras is an open-source Python library for developing and evaluating ML models [7]. We also use the sklearn library to evaluate our ML models' performance since it is a simple and efficient tool for predictive data analysis [19].

6.1 Deep Neural Networks

A Deep Neural Network (DNN) is an Artificial Neural Network (ANN) that consists of multiple hidden layers between the input and output layers [5]. Our proposed DNN model has one input layer, three hidden layers, and one output layer. The input layer has four nodes to handle the feature matrix, the first, second, and third hidden layer have 50, 50, and 20 nodes respectively and finally one node forms the output layer. We select these model parameters empirically.

We use a *ReLU* (Rectified Linear Unit) activation function in the hidden layers for better performance. Using the *ReLU* function, all the neurons are not active at the same time. The output layer uses a *Sigmoid* activation function to ensure the model output is in the range (0,1) and later map to one of two classes (attack or non-attack) with a default threshold of 0.5.

We train our DNN model with the following parameters: The number of epochs is 150. We use *binary_crossentropy* as our loss function. It is a cross entropy loss function defined in Keras which is used for binary classification problems. We also use *adam* as our model optimizer. This optimizer is an efficient stochastic gradient descent algorithm. It is a self-tuned version of gradient descent. All these parameters are selected based on our experimental results.

6.2 Support Vector Machines

Support vector machines are one of the supervised learning methods used for classification. SVM aims to find a hyper-plane in vector space (or a line in 2D space) to classify data points into two binary classes [1]. It produces significant accuracy with low computation power which is a good match for IoT networks.

To separate data into two classes, several hyper-planes could be selected. Our aim is to find the one that maximizes the margin between the two classes. SVM uses closer data points to the hyper-plane named support vectors, to find the shape and position of the hyper-plane. These support vectors help to maximize the margin of the classifier.

SVM uses a technique called kernel-trick to transform the data points and create an optimal decision boundary between the classes. In this study we use the two main kernels, i.e., Radial Basis Function (RBF) and Polynomial Function. RBF is one of the most preferred and used kernel functions in SVM. It works well for non-linear data. It also helps to properly classify the points when there is no prior knowledge of data.

6.3 Random Forest Classifiers

We also use RFC, an ensemble learning method using Decision Trees. Random forest algorithms have been popular in embedded devices because of their simple nature, performance, low memory requirements, and since they are less susceptible to over-fitting [26, 1].

6.4 Training Set

After data labeling, we split our data into train, validation and test sets. The train set is used for training the ML models. We use the validation set to tune the model's parameters. Finally, the test set is used to evaluate the performance of ML models with an unseen dataset. We split the labeled dataset into three sets of train (60%), validation (20%), and test (20%). The ratio is widely used in ML problems and works well when we have enough labeled data.

7 Evaluation

7.1 Metrics

We measure the ML models' performance using the test data set which includes unseen data. The two main metrics we use are prediction accuracy and precision.

Prediction Accuracy. Prediction accuracy is the number of correct predictions made by the model divided by all the predictions made (Equation 2).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2}$$

Precision. Precision is a measure that defines what proportion of nodes that we diagnosed as under attack, actually was under attack. The predicted positives (nodes that are predicted as under attack are TP and FP) and the nodes that are under attack are TP (Equation 3). Precision gives us information about the model's performance with respect to false positives. In other words, it shows that how successful is an ML model to catch the attacks.

$$Precision = \frac{TP}{TP + FP} \tag{3}$$

7.2 Basic Model Performance

Table 3 compares the accuracy and precision of the DNN, RFC, and SVM-RBF models. The table shows that DNN and SVM with RFB kernel have almost identical accuracy (97%) and precision of (95%) and (96%) respectively. The highest accuracy is achieved by the RFC model with 98% accuracy and 96% precision. The proposed ML models have different complexity. DNN and SVM need more resources for the training part, so we need train our ML model on the edge or the cloud. On the other hand, RFC models have less complexity to run on resource constraint low-power IoT mesh networks.

7.3 Number of Chunks

Figure 6 shows the accuracy difference between using different numbers of chunks (see Section 5.3).

The experimental results show that for DNN and RFC, six chunks perform better than three and 10 chunks.

Table 3. Performance of machine learning models using six chunks

Model Type	Accuracy (%)	Precision (%)
DNN	97	95
RFC	98	96
SVM-RBF	97	96

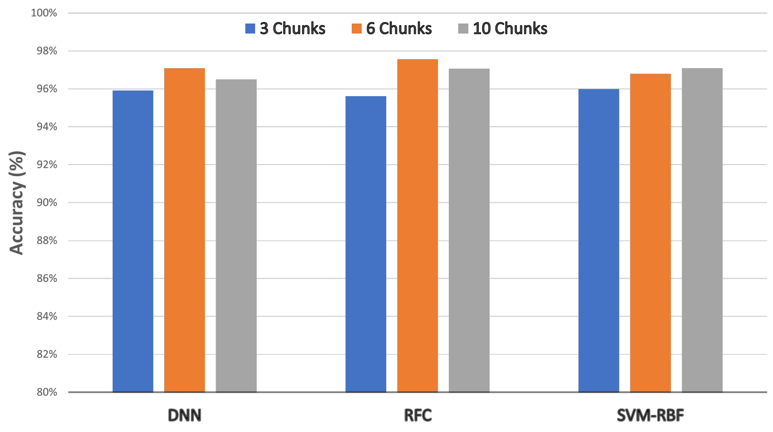


Fig. 6. The number of chunks and its effect on the detection accuracy

7.4 Generalization

To understand how well our models generalize, we compare the ML models performance using different training sets. For this, we train our models with datasets that include more and diverse data, i.e., the extended data trace includes new topologies with different number of nodes. We use six chunks in these experiments since this leads to good performance. Our test set consisting of topologies with five to 20 nodes (10 different topologies for each number of nodes). Figure 7 demonstrates that by adding new data traces from new scenarios, the accuracy of the ML models increases. For example, the accuracy of SVM-RBF increases from about 92% when trained with scenarios consisting of five nodes to more than 96% when training with scenarios consisting of five to 20 nodes.

We also conduct an experiment in which we train our ML models with scenarios consisting of 5 to 20 nodes and then test with an unseen dataset of scenarios of 30 nodes, i.e., the size of the networks in the test set is larger than in the train set. Even in this scenario, DNN and RFC, SVM-RBF achieve an accuracy of 95% confirming that our models generalize well to unseen topologies.

Overall, these experiments show that our approach to extend the dataset with new topologies with a different number of nodes leads to ML models that detect attacks with higher accuracy.

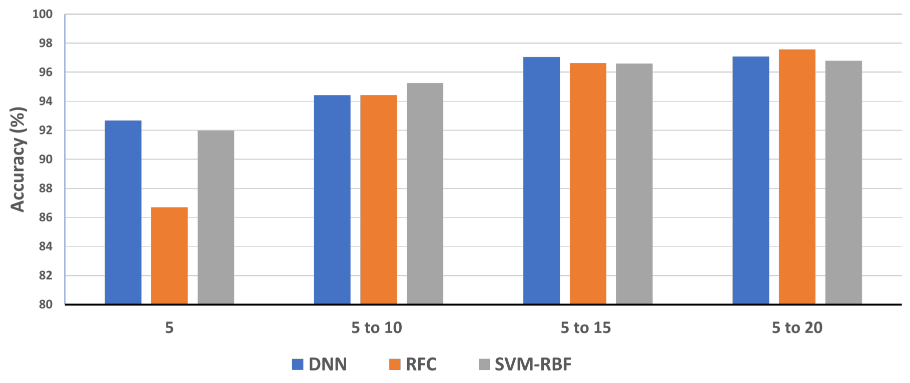


Fig. 7. The ML models' accuracy increases when we train with more and more diverse data.

8 Conclusions

Due to the resource-constraints of the devices and their wireless nature, IoT mesh networks are exposed to many attacks. Previous studies on intrusion detection systems for such networks, have usually trained their machine learning algorithms on small datasets, often with a fixed number of nodes. In this paper, we have shown that using larger training datasets that are also more diverse, has a positive affect on model accuracy. Our results using the blackhole attack on the RPL routing protocol have shown that we achieve high accuracy on unseen test data even when the size of the networks is larger in the test than in the training set.

The generated new attack dataset for IoT multi-hop networks can be found at the following repository at Github: [https://github.com/STACK-ITEA-Project/multi-trace-data/raw/master/data/data_attack_blackhole.zip]

Acknowledgments

This work was supported by the ITEA 3 project STACK funded by the Swedish Innovation Agency VINNOVA.

References

1. Ahmad, I., Basheri, M., Iqbal, M.J., Rahim, A.: Performance comparison of support vector machine, random forest, and extreme learning machine for intrusion detection. IEEE access **6**, 33789–33795 (2018)
2. Alabsi, B.A., Anbar, M.: A comprehensive review on security attacks in dynamic wireless sensor networks based on rpl protocol (2018)
3. Ambili, K., Jose, J.: Tn-ids for network layer attacks in rpl based iot systems. IACR Cryptol. ePrint Arch. **2020**, 1094 (2020)
4. Ashton, K., et al.: That 'internet of things' thing. RFID journal **22**(7), 97–114 (2009)
5. Bengio, Y.: Learning deep architectures for AI. Now Publishers Inc (2009)
6. Butun, I., Morgera, S.D., Sankar, R.: A survey of intrusion detection systems in wireless sensor networks. IEEE communications surveys & tutorials **16**(1), 266–282 (2013)
7. Chollet, F., et al.: Keras. https://github.com/fchollet/keras (2015)
8. Diro, A.A., Chilamkurti, N.: Distributed attack detection scheme using deep learning approach for internet of things. Future Generation Computer Systems **82**, 761–768 (2018). https://doi.org/https://doi.org/10.1016/j.future.2017.08.043, https://www.sciencedirect.com/science/article/pii/S0167739X17308488
9. Dunkels, A., Grönvall, B., Voigt, T.: Contiki - a lightweight and flexible operating system for tiny networked sensors. In: Proceedings of the IEEE Workshop on Embedded Networked Sensor Systems (IEEE Emnets). Tampa, Florida, USA (Nov 2004)
10. Essop, I., Ribeiro, J.C., Papaioannou, M., Zachos, G., Mantas, G., Rodriguez, J.: Generating datasets for anomaly-based intrusion detection systems in iot and industrial iot networks. Sensors **21**(4), 1528 (2021)
11. Finne, N., Eriksson, J., Voigt, T., Suciu, G., Sachian, M.A., Ko, J., Keipour, H.: Multi-trace: Multi-level data trace generation with the cooja simulator. In: Workshop on Machine Learning for Smart Wireless Networks (2021)
12. Karie, N.M., Sahri, N.M., Haskell-Dowland, P.: Iot threat detection advances, challenges and future directions pp. 22–29 (2020)
13. Liang, C., Shanmugam, B., Azam, S., Karim, A., Islam, A., Zamani, M., Kavianpour, S., Idris, N.B.: Intrusion detection system for the internet of things based on blockchain and multi-agent systems. Electronics **9**(7), 1120 (2020)
14. Moustafa, N., Hu, J., Slay, J.: A holistic review of network anomaly detection systems: A comprehensive survey. Journal of Network and Computer Applications **128**, 33–55 (2019)
15. Nikoukar, A., Raza, S., Poole, A., Güneş, M., Dezfouli, B.: Low-power wireless for the internet of things: Standards and applications. IEEE Access **6**, 67893–67926 (2018)
16. Nobakht, M., Sivaraman, V., Boreli, R.: A host-based intrusion detection and mitigation framework for smart home iot using openflow. In: 2016 11th International conference on availability, reliability and security (ARES). pp. 147–156. IEEE

17. Österlind, F., Dunkels, A., Eriksson, J., Finne, N., Voigt, T.: Cross-level sensor network simulation with cooja. In: Proceedings of the First IEEE International Workshop on Practical Issues in Building Sensor Network Applications (SenseApp 2006). Tampa, Florida, USA (Nov 2006)
18. Pasikhani, A.M., Clark, J.A., Gope, P., Alshahrani, A.: Intrusion detection systems in rpl-based 6lowpan: A systematic literature review. IEEE Sensors Journal (2021)
19. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)
20. Phukan, J., Li, K.F., Gebali, F.: Hardware covert attacks and countermeasures. In: 2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA). pp. 1051–1054. IEEE (2016)
21. Pongle, P., Chavan, G.: A survey: Attacks on rpl and 6lowpan in iot. In: 2015 International conference on pervasive computing (ICPC). pp. 1–6. IEEE (2015)
22. Samaila, M.G., Neto, M., Fernandes, D.A., Freire, M.M., Inácio, P.R.: Challenges of securing internet of things devices: A survey. Security and Privacy **1**(2), e20 (2018)
23. Sharma, M., Elmiligi, H., Gebali, F., Verma, A.: Simulating attacks for rpl and generating multi-class dataset for supervised machine learning. In: 2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON). pp. 0020–0026. IEEE (2019)
24. Sharma, S., Gupta, R.: Simulation study of blackhole attack in the mobile ad hoc networks. Journal of Engineering Science and Technology **4**(2), 243–250 (2009)
25. Verma, A., Ranga, V.: Evaluation of network intrusion detection systems for rpl based 6lowpan networks in iot. Wireless Personal Communications **108**(3), 1571–1594 (2019)
26. Wang, H., Barriga, L., Vahidi, A., Raza, S.: Machine learning for security at the iot edge-a feasibility study. In: 2019 IEEE 16th International Conference on Mobile Ad Hoc and Sensor Systems Workshops (MASSW). pp. 7–12. IEEE (2019)
27. Xu, J., Sun, X., Zhang, Z., Zhao, G., Lin, J.: Understanding and improving layer normalization. arXiv preprint arXiv:1911.07013 (2019)
28. Zarpelão, B.B., Miani, R.S., Kawakani, C.T., de Alvarenga, S.C.: A survey of intrusion detection in internet of things. Journal of Network and Computer Applications **84**, 25–37 (2017)