A COLLABORATIVE NETWORK INTRUSION DETECTION ARCHITECTURE FOR A PROGRAMMABLE DATA PLANE

Thi Nga Dao[1,*], *Manh Hung Tran*[1], *Van Duc Le*[2]

doi:10.56651/lqdtu.jst.v11.n02.536.ict

Abstract

For early detection and response to network threats, a network intrusion detection system should be executed on a data plane. However, due to high model complexity, an intrusion detection model based on advanced machine learning techniques becomes unsuitable for limited-resource switches. To address this problem, we propose a lightweight joint detection model that is inspired by classification parallelism and neuron pruning. Specifically, the traditional multi-label classification model is decoupled into several class-specific sub-models and each sub-model takes charge of detecting one or several traffic classes. In our model, the number of participating switches can vary based on network traffic and available computing resources of edge devices. Moreover, to reduce the size of sub-models, magnitude pruning is applied for each sub-model to only keep salient connections. Evaluation experiments are conducted with various network parameters and results show that the proposed architecture achieves much lower model complexity than the traditional multi-label classifier without a reduction in classification performance.

Index terms

Traffic management, intrusion detection, neuron pruning

1. Introduction

A network intrusion detection system (NIDS) has received great attention from researchers and practitioners since it plays a main role in strengthening the network security [1]–[3]. NIDS can be located in an external device that collects packet information and predict possible attacks [4]–[6]. However, using the external device leads to significantly high detection delay. Therefore, NIDS should be distributed at edge devices (e.g., networking switches) to achieve quick detection time and early response to network threats. When deploying NIDS in a data plane, the lack of computing resources of edge devices should be considered carefully, especially when the data rate is high. Although a lot of intrusion detection systems were proposed in the literature, there is still a lack of detection models on distributed edge devices. To address these issues, we

[1]Faculty of Radio-Electronic Engineering, Le Quy Don Technical University, Vietnam
[2]School of Computer Science and Engineering, Nanyang Technological University, Singapore
[*]Corresponding author: daothinga@mta.edu.vn

propose a collaborative network intrusion detection model that allows multiple switches to participate in the detection task.

To fast inference on online data items, Wang *et al.* introduced class parallelism called SensAI in which a single neural network (NN) is decoupled into multiple binary classifiers or sub-models [7]. Each class-specific classifier consists of important neurons for a certain type of class and these sub-models can run over multiple machines simultaneously and independently. A softmax layer is applied after combining outputs of the sub-models to obtain the classification result. Due to the smaller size of each sub-model than the original network, SensAI achieves a huge reduction in model complexity and inference time, as shown in the CIFAR10 dataset.

Inspired by class parallelism and magnitude pruning, we design NIDS that can be embedded in a chain of resource-constrained switches. The collaborative detection architecture is divided into multiple binary sub-models and each of these sub-models is implemented in a single device to detect one or several attack classes. The main difference between SensAI and our model is that SensAI needs N devices in the case of N classes and each device takes responsibility for a specific traffic class. Meanwhile, our model can deploy on M devices ($M \leq N$) and a device can detect one or some classes. A threshold value λ ($0 \geq \lambda \geq 1$) is used to determine whether an incoming packet is classified at a device. More specifically, if the probability of a traffic class is greater than λ, we can conclude the traffic type of the incoming packet without requiring other devices. Otherwise, another device is needed for traffic inspection. In the worst case, we combine outputs of the sub-models and add a softmax layer at the last device to conclude the traffic label. The complexity of each sub-model is smaller than that of the original multi-label architecture. To further reduce model complexity, we adopt magnitude pruning for sub-models after training parameters of these sub-models.

There are three steps for model training. First, network parameters of fully-connected class-specific sub-models are trained separately by minimizing an entropy-based loss function. Then, we apply magnitude pruning for sub-models independently. Specifically, connections with the lowest absolute weight values are removed from the fully-connected models. Finally, the whole collaborative architecture is fine-tuned after adding a softmax layer, which merges outputs of sub-models. An entropy-based loss function is used to find the optimal parameters. For performance evaluation, the IoT dataset [8] with nearly three million samples and five attack types is used. Main performance metrics include accuracy, confusion matrix, and the number of floating-point operations (FLOPs).

The main contributions of our work is summarized as follows:

- For model complexity reduction, we propose a lightweight distributed detection architecture based on class parallelism and a neuron pruning method. The proposed architecture can run on multiple constrained-resource switches and the number of switches can be lower than the number of attack classes.
- We evaluate the proposed joint detection model with different parameters including pruning rate, the number of hidden neurons, the number of participating switches,

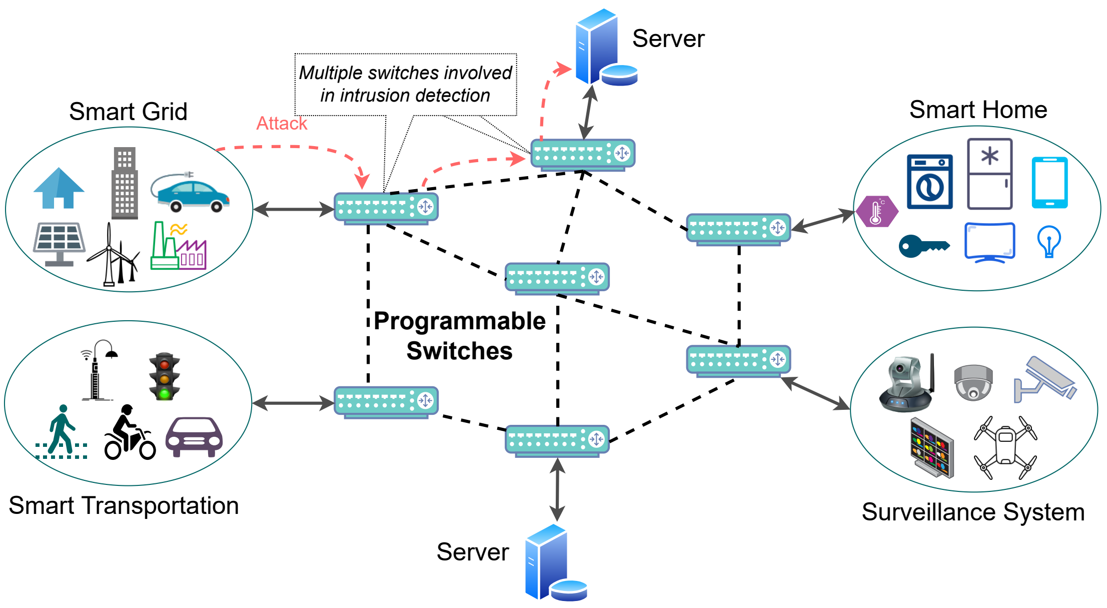


Fig. 1. The overview of NIDS in Networks.

and a threshold value.

- Experimental results show that the proposed distributed model achieves low model complexity with similar classification performance to the traditional multi-label classifier.
- We find the tradeoff between classification accuracy and model complexity when choosing network parameters such that pruning rate, threshold value, and the number of participating devices.

The rest of the paper is organized as follows: Section 2 presents the model system and assumptions. Then, the proposed collaborative detection model is described in Section 3. Section 4 shows the evaluation performance of the proposed model with various network parameters. Finally, we draw the conclusion for our work in Section 5.

2. Network system

We assume that switches are used to forward data traffic generated from a device to a destination. Data traffic can be from a variety of applications such that smart grid, smart transportation, smart home, and surveillance system. As shown in Figure 1, these switches are connected using an arbitrary network topology: bus, star, tree, ring, or mesh. Attacks can be injected into traffic at any points from the generated device to the destination host. To ensure the network security, data traffic should be inspected at edge devices.

When a data packet arrives at the input port, the switch makes a prediction on the traffic type of the packet. In the case of the IoT dataset [8], there are five different traffic labels: normal, reconnaissance, man-in-the-middle, denial-of-service, and botnet. Based on the prediction output, the switch can take appropriate action for this packet, e.g., forwarding the packet, dropping the packet, or adding an alarm field in the header.

For quick detection and prevention of network threats, we execute the traffic classification model on programmable data plane. Data plane programmability allows customization of packet processing functions on edge devices, thus leading to a lower detection delay than the case of sending data traffic to an external device for examination. Several commercial programming edge devices including NetFPGA SUME developed by Digilent [9] and Intel Tofino2 [10].

A common architecture of a programmable switch includes four main blocks: parser, ingress control, outgress control, and deparser. When an incoming packet arrives at the switch, the Ethernet, IP, and TCP headers are analyzed to extract necessary information (e.g., source and destination IP addresses, source and destination port numbers, and arriving time). This information is used to derive input features of the proposed architecture. Then, the model outputs the probability of occurring attacks. The packet is processed at the incoming and outgoing ports by ingress and outgress control blocks, respectively. Based on the probability of attack types, look-up tables are used to find appropriate actions (e.g., sending data to a specific port and updating the time-to-live parameter) for the given packet. The switch can select one of the possible actions after packet examination.

To embed the intrusion detection function on the programmable switch, we use the P4 programming language [11], [12]. Since P4 is a domain-specific language, it only supports a limited number of operations for binary and integer numbers. Arithmetic operations including addition, subtraction, and multiplication are supported while no division/modulo operation can be used. Bit shift and element-wise comparison operations are also supported. We apply the right bit shift instead of division operation to control the number of bits used for units in the network.

When designing an intrusion detection function on a data plane, we should consider the fact that edge devices are equipped with limited computing resources and memory footprints. Therefore, it is a need for a lightweight distributed architecture that can make use of resources of multiple switches for intrusion detection. The next section presents a collaborative detection model followed by a performance evaluation.

3. Collaborative intrusion detection on multiple distributed switches

For early detection and quick response to network threats, NIDS should be implemented in the data plane. However, since edge devices are usually equipped with limited resources and memory footprints, the detection architecture should have low model complexity. In this work, we develop a lightweight intrusion detection model that allows joint detection between multiple networking devices. To do that, the idea of class parallelism [7] is modified in which the traditional multi-label classification model is decoupled into numerous sub-models with lower model complexity than the original

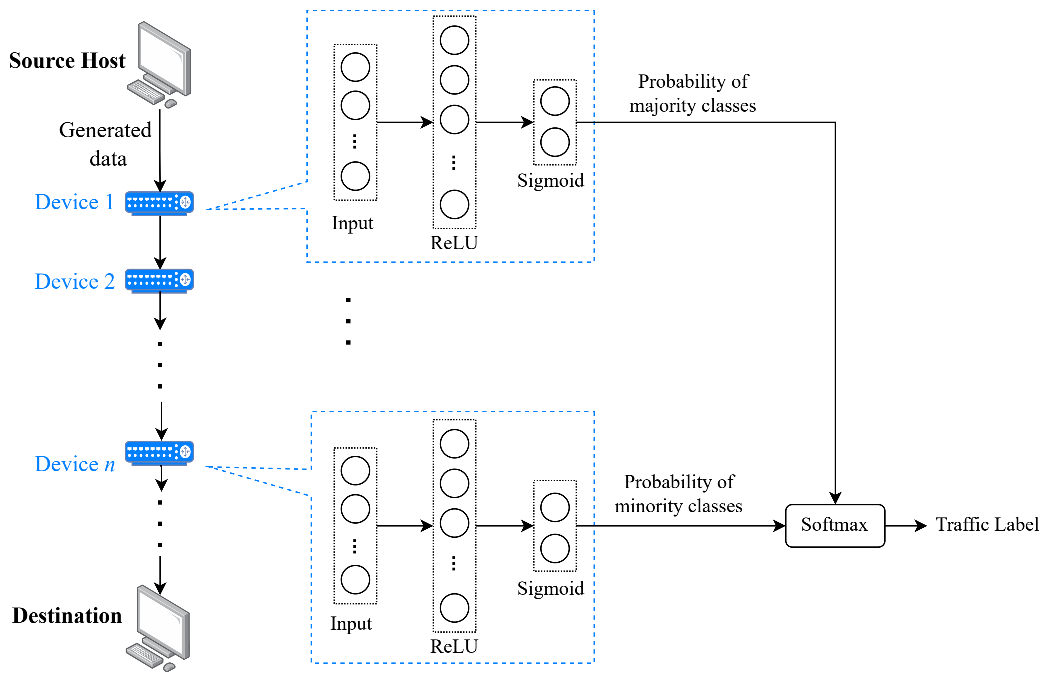


Fig. 2. The joint intrusion detection architecture.

one. Each sub-model is implemented at a device to output the probability of one or multiple attack types. To more lessen the number of operations in each sub-model, a neuron pruning algorithm is utilized to remove unnecessary connections. In this section, we first present the main architecture of the joint detection model and how to train the network parameters of the proposed model.

Let n_{label} denote the number of traffic labels in which the security system is interested. As shown in Figure 2, the proposed architecture decomposes a detection function into k class-specific sub-models ($k \leq n_{label}$). Let $n_1, n_2, ...,$ and n_k denote the number of attacks of which switches 1, 2, ..., and k take in charge, respectively ($\sum_{i=1}^{k} n_i = n_{label}$). When packets arrive at device 1, data features are extracted and used to predict the probability of the major attack classes. The architecture of the sub-models consists of three layers: input, hidden, and output layers. Since the P4 programming language only supports operations for integers and binary, the ReLU function is used at the hidden layers. Meanwhile, because sub-models return the probabilities of attacks, the sigmoid function, which outputs a value between 0 and 1, is considered at the output layer of sub-models. At the last device, a softmax layer is added to determine the most likely traffic type. Packet features are only extracted at the first device and then these features are sent to other devices.

To construct the joint detection model, three main steps are required as shown in Figure 3. Before nework training, we need to know the training dataset and hyper-

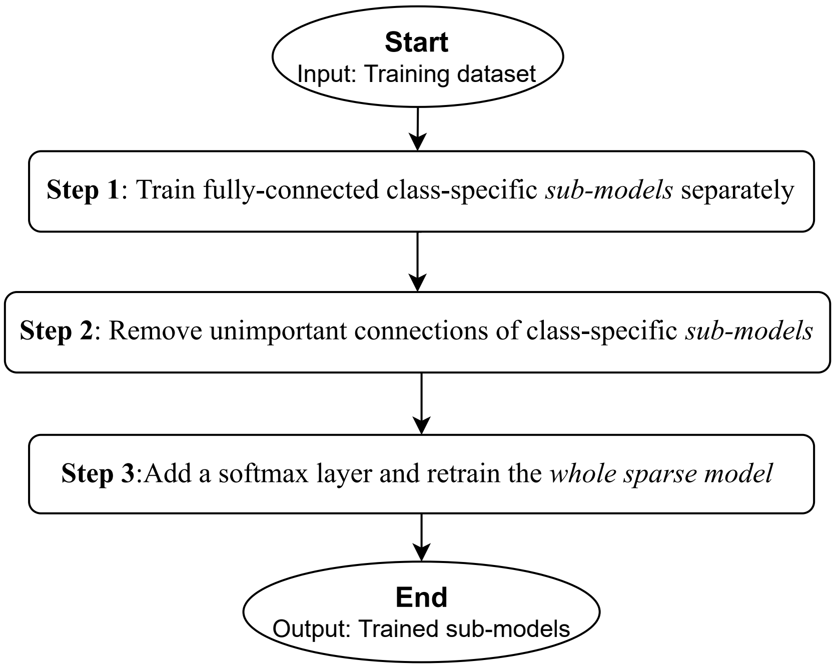

Start
Input: Training dataset
Step 1: Train fully-connected class-specific sub-models separately
Step 2: Remove unimportant connections of class-specific sub-models
Step 3:Add a softmax layer and retrain the whole sparse model
End
Output: Trained sub-models

Fig. 3. Training procedure.

parameters such as threshold value and pruning rate. The purpose of the training procedure is to find learned parameters including weights and biases. Parameters training is implemented at a centralized device and then learnt parameters are sent to distributed switches. In step 1, we train fully-connected sub-models independently by minimizing an entropy-based loss function. The loss function L_i at device i is defined as below.

$$L_i = -\frac{1}{m}\sum_{j=1}^{m} t_i^{(j)}\log(y_i^{(j)}) \tag{1}$$

where m is the number of training samples while $t_i^{(j)}$ and $y_i^{(j)}$, respectively, are the target and predicted values of the j^{th} sample at device i. Training terminates when there is no improvement in classification accuracy on the validation set for the most recent 20 epochs.

In the next training step, we apply magnitude pruning in which links with the lowest absolute weight values in sub-models are totally removed. Since trained weight values of sub-models may have a big gap, connection removal should be done separately between sub-models to avoid layer collapse. Layer collapse is a situation when there is no remaining connection between two consecutive layers, which makes the network untrainable. The amount of trimmed connections depends on pruning rate, which is denoted by p_{prune} $(0 \leq p_{prune} \leq 1)$. For example, if $p_{prune} = 0.3$ and there are 100 connections in a fully-connected sub-model, we remove 30 weakest links and keep 70

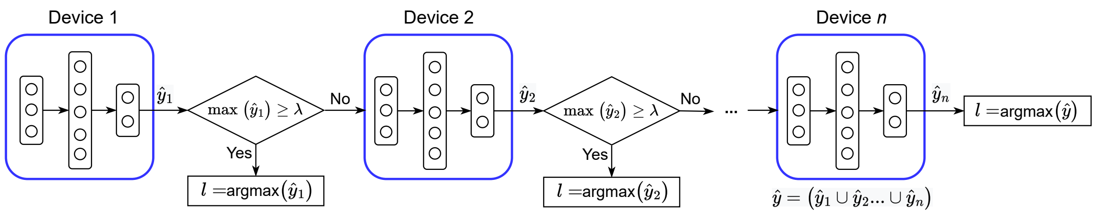

Device 1
Device 2
Device n
$\hat{y}_1$
$\max(\hat{y}_1) \geq \lambda$
No
Yes
$l = \text{argmax}(\hat{y}_1)$
$\hat{y}_2$
$\max(\hat{y}_2) \geq \lambda$
No
Yes
$l = \text{argmax}(\hat{y}_2)$
$\hat{y}_n$
$l = \text{argmax}(\hat{y})$
$\hat{y} = (\hat{y}_1 \cup \hat{y}_2 ... \cup \hat{y}_n)$

Fig. 4. Inference phase.

strongest connections. A binary mask matrix M with the same size as the weight matrix to denote the pruning status of weights. Values 0 and 1 represent removed and remaining connections, respectively. Since sub-models are trained and pruned separately, we can make use of multiple processors in the first two steps to speed up the training process.

In the final step, a softmax layer is added by combining outputs of sub-models to determine the most likely traffic label. We minimize an entropy-based loss function L that is derived as follows.

$$L = -\frac{1}{m}\sum_{j=1}^{m}\sum_{i=1}^{n}\mathbf{t}_i^{(j)}\log(\mathbf{y}_i^{(j)}) \tag{2}$$

where $\mathbf{t}_i^{(j)}$ is the target value of the i^{th} training sample and $\mathbf{y}_i^{(j)}$ is the output of the softmax layer. At each epoch, the remaining parameters of sub-models are updated.

Figure 4 shows how the proposed joint detection model classifies an online packet. As a packet arrives at device 1, the first sub-model is executed to compute the probabilities $\hat{y}_1$ of n_1 attack types. If the maximum value of $\hat{y}_1$ is greater than a threshold value, which is denoted by λ, the incoming packet is classified at device 1 and classification for this packet is done. Otherwise, packet features are sent to device 2 in which the sub-model 2 is implemented to derive the probabilities of the next n_2 traffic labels. Similarly, if the condition $max(\hat{y}_2) \geq \lambda$ is satisfied (i.e., the system is quite sure about the traffic class), we can conclude the attack label at device 2 without sending packet features to next devices. Otherwise, another device is involved in packet classification. In the worst case, the process is repeated until device n, which outputs the probabilities of the last n_k attack classes and combines outputs of all sub-models to return the most probable traffic type.

4. Experimental results

In this section, we present results for evaluating the proposed joint detection model and comparing it with existing work. A desktop PC with Intel Core i7 2.5GHz CPU (with the Radeon R9 M370X 2048 MB and Intel Iris Pro 1536 MB GPU support) and 16 GB RAM is used to conduct experiments. Table 1 shows attack distribution of the IoT dataset [8] with five different traffic classes: normal, denial-of-service, man-in-the-middle, botnet, and reconnaissance. There are several reasons for selection of this dataset. Firstly, samples in the IoT dataset are collected from real network experiments in a smart home environment. Secondly, this dataset was published in 2019 and quite up-to-date. Finally, the dataset provides five different traffic classes and the total number of samples in the dataset is nearly three millions, which is big enough to train a machine learning model. The whole dataset is split into training, validation, and test sets with a ratio of 5:2:3. The training and validation sets are used during the training process while the test set is for performance evaluation. In this paper, we use the number of traffic samples in each class to distinguish between majority and minority classes. More specifically, traffic classes including normal, botnet, and Mitm with the highest number of samples in the Korea IoT dataset are selected as majority classes. The remaining classes are called minority classes. Six salient input features from the set of features provided in [13] are used for experiments. The IoT dataset only provides raw packet trace at devices. To derive input features of the classification model, whenever a packet arrives at the switch, the source and destination IP addresses are extracted from the packet header. The selected features include weight, mean, and variance of packet length with the flow id is the source IP address and weight, mean, and variance of the elapsed period between two consecutive arriving times with the flow id is the source and destination IP addresses. Note that these features are computed statistically by switches and presented using a 32-bit binary number [14]. Different learning rates are examined and the model with the highest performance on the validation set is evaluated and presented in this paper. Table 2 summarizes key parameters used in experiments. Default values are highlighted with bold texts.

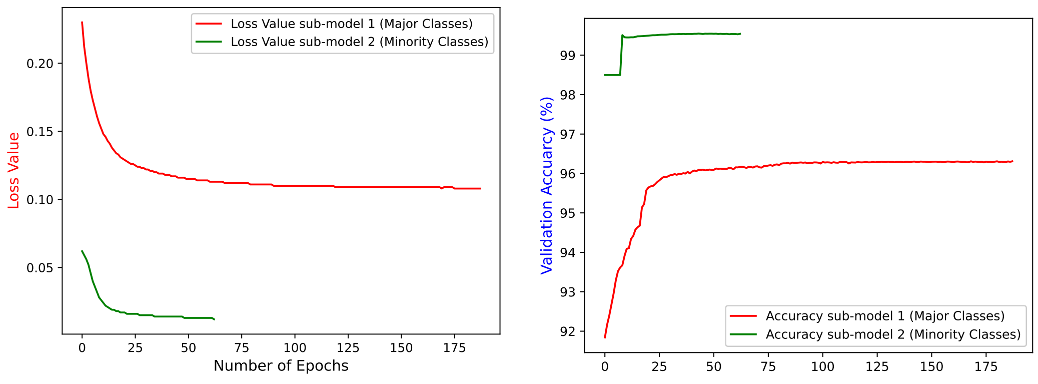
Table 1. Attack distribution of the IoT network intrusion dataset.

Name	**No. of packets**	**Percentage** (%)
Normal	1,756,276	58.82
Botnet	1,037,977	34.76
MitM	101,885	3.41
DoS	64,646	2.16
Reconnaissance	25,210	0.84
Total	**2,985,994**	**100**

First, we consider the situation when two switches are involved into attack detection. The first switch takes in charge of classifying three major attacks: Normal, Botnet, and MitM. Meanwhile, the remaining two classes are for the second switch. Figure 5 presents learning curves during the training process. The number of hidden neurons is set to ten and there is one hidden layer in each sub-model. The loss function decreases and the accuracy of both sub-models improves when there are more training epochs. When comparing performance between two sub-models, we observe that the second sub-model achieves higher classification accuracy than the first sub-model. Specifically, after the first training phase, the sub-model 1 produces around 96% accuracy compared to

Table 2. Parameters setup.

Parameter	**Value**
The number of devices	$\{\mathbf{2}, 3, 4, 5\}$
#Hidden layer	1
#Hidden Neurons	$\{5, \mathbf{10}, 20, 30\}$
Threshold Value (λ)	between 0.01 to 0.999
Learning rate	$\{0.001, 0.003, 0.005, 0.01, 0.03, 0.05\}$
Optimizer	Adam optimizer [15]
p_{prune}	from 0.1 to 0.9

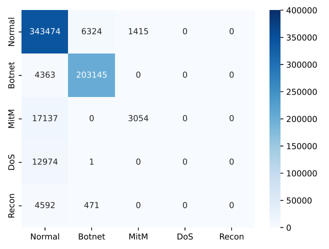


(a) (b)

Fig. 5. Learning curves on the validation set during the first training step: (a) Loss value and (b) Classification accuracy.

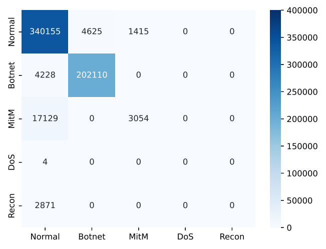
nearly 100% of the sub-model 2. One factor contributing to this observation is that both sub-models have ten hidden units but the sub-model 1 needs to classify three classes compared to two classes of the sub-model 2. Therefore, the sub-model 1 should have a more complex architecture than the sub-model 2. It can also be inferred from Figure 5 that the first sub-model needs more training epochs than the second one (188 compared to only 63).

We examine the accuracy and detection delay with different threshold values from 0.01 to 0.999 as shown in Figure 7. Recall that the threshold value λ is used to determine whether a packet can be classified before arriving at the last device. More specifically, if the maximum output unit of a sub-model exceeds λ, the incoming packet is predicted to belong to a specific label. If using a high threshold value, we are more sure about label prediction. As a result, when using a larger threshold value, classification accuracy can significantly increase. Performance becomes convergent with $\lambda \geq 0.3$ for all values of hidden units. Note that there is more likely that packets are classified by the second switch instead of the first switch in the case of higher λ. As a result, the detection delay, which includes transmission latency between two devices, becomes much larger. Therefore, it is important to select an optimal threshold value to reduce the detection



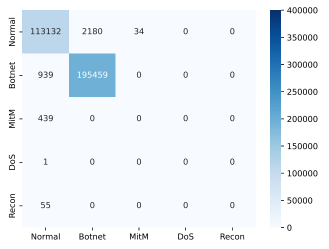
	Normal	Botnet	MitM	DoS	Recon
Normal	343474	6324	1415	0	0
Botnet	4363	203145	0	0	0
MitM	17137	0	3054	0	0
DoS	12974	1	0	0	0
Recon	4592	471	0	0	0

(a) λ = 0.1



	Normal	Botnet	MitM	DoS	Recon
Normal	340155	4625	1415	0	0
Botnet	4228	202110	0	0	0
MitM	17129	0	3054	0	0
DoS	4	0	0	0	0
Recon	2871	0	0	0	0

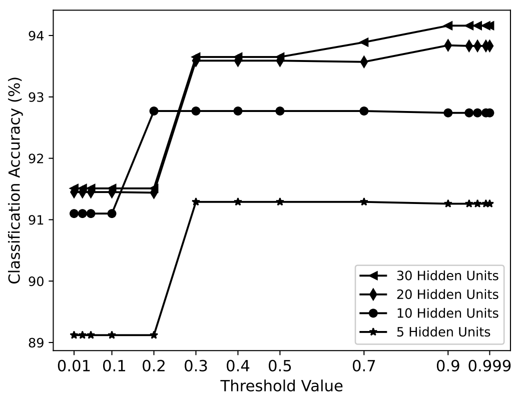
(b) λ = 0.5



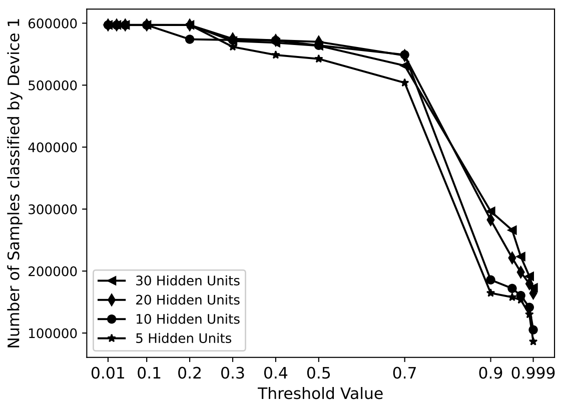
	Normal	Botnet	MitM	DoS	Recon
Normal	113132	2180	34	0	0
Botnet	939	195459	0	0	0
MitM	439	0	0	0	0
DoS	1	0	0	0	0
Recon	55	0	0	0	0

(c) λ = 0.9

Fig. 6. Confusion matrix of samples classified by the sub-model 1 with different threshold values: a) λ = 0.1, b) λ = 0.5, c) λ = 0.9.



(a)



(b)

Fig. 7. Impact of threshold value on performance: (a) Classification accuracy and (b) Detection delay.

delay. The number of hidden features is set from five to 30 and there is one hidden layer. When we increase the number of hidden units, classification performance can be clearly improved but the model size increases. Therefore, a network with ten hidden neurons are selected as the default architecture for a balance between accuracy and model size.

To better presentation of impact of λ, Figure 6 shows confusion matrices of samples examined by the first sub-model when λ is set to {0.1, 0.5, 0.9}. The total number of test samples is 597,200. The sub-model 1 classifies 596,950; 575,591; 312,239 samples while the accuracy of the sub-model 1 is 91.6%, 94.2%, 98.8% with $\lambda = 0.1$, 0.5, and 0.9, respectively. That means prediction by the sub-model 1 becomes more accurate when we increases the threshold value. However, fewer samples can be predicted at the sub-model 1 in the case of a larger λ value.

Next, the impact of the pruning rate on classification performance is shown in Figure 8 in terms of accuracy and the number of FLOPs including summation and multiplication. The pruning rate is set from 0.1 to 0.9 with the step size being 0.1. The number of FLOPs is derived as: $2(n_x n_h + n_h n_y)(1 - p_{prune})$ where n_x, n_h,and n_y denote the number of input, hidden, and output neurons. Generally, classification accuracy deteriorates and the model complexity is reduced when we use a high pruning rate value. Since fewer connections are used to distinguish between attack types with a high pruning rate, there is a decreasing trend in classification performance. There is a balance between accuracy and model complexity. If high accuracy is required, the model should be more complex and vice versa. In this work, the pruning rate of 0.6 is selected as the default value for other experiments to achieve both relatively high accuracy and low model complexity.

We also compare the performance of the proposed joint distributed architecture and the centralized detection method in which the multi-label classification model is implemented on a single device [14]. As shown in Figure 8, our joint detection model achieves similar accuracy to the centralized model while reducing the number of FLOPs. In terms of classification accuracy, the centralized method has low performance when p_{prune} is set to 0.9. The reason is that some output units are isolated from the previous layer, that means these units could not be updated during the re-training and inference phases. The isolation problem can occur when applying a weight pruning method, especially with a high pruning rate and a sparse network. Since the fully-connected centralized model has a lower number of connections than the joint detection model, the centralized model is more vulnerable to the isolation problem than the proposed architecture. Since the magnitude pruning considers removing weights, i.e., connections between neurons, the probability of the isolation problem is higher than the case of neuron pruning. Therefore, to improve the classification performance of the defense system, we can use a more advanced neuron pruning method such as layer-wise relevance propagation (LRP) pruning [16]. In LRP, scores of hidden neurons at a specific layer are computed based on scores of the activation units of the succeeding layer and weights connecting these layers. Table 3 shows the improvement of LRP over the magnitude pruning method. Thanks to more careful consideration of hidden neuron contribution to the output layer, the improvement is clearly shown with a larger pruning rate.

Table 3. Performance comparison between the magnitude pruning and LRP method.

Pruning rate	**Magnitude pruning**	**LRP**
0.1	94.27%	94.14%
0.3	93.78%	93.69%
0.5	93.35%	93.81%
0.7	91.79%	93.05%
0.9	87.66%	92.69%
0.95	86.92%	92.33%

Figure 9 presents the change in classification accuracy when the number of collaborative devices varies from two to five with different threshold values. Note that there is latency for packet transmission between participating switches. Therefore, if more devices are involved in traffic classification, detection delay becomes much longer. As a result, the number of switches should be as low as possible to avoid a long classification delay. As shown in Figure 9, when there are two switches, relatively high accuracy can

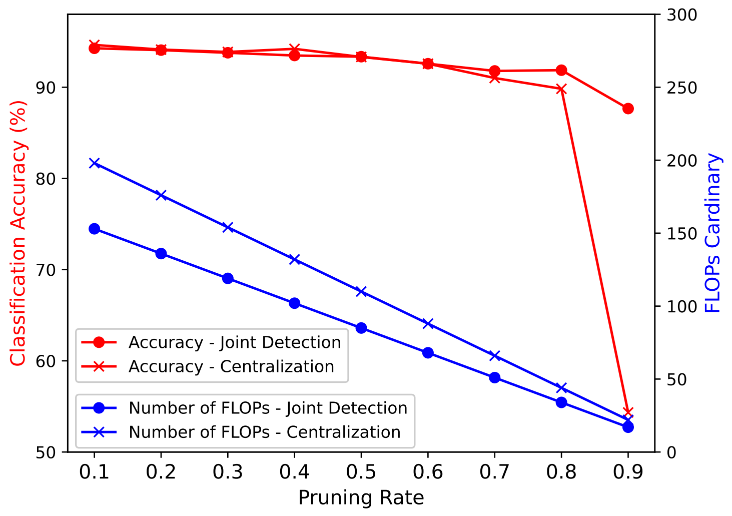


Fig. 8. Impacts of pruning rate.

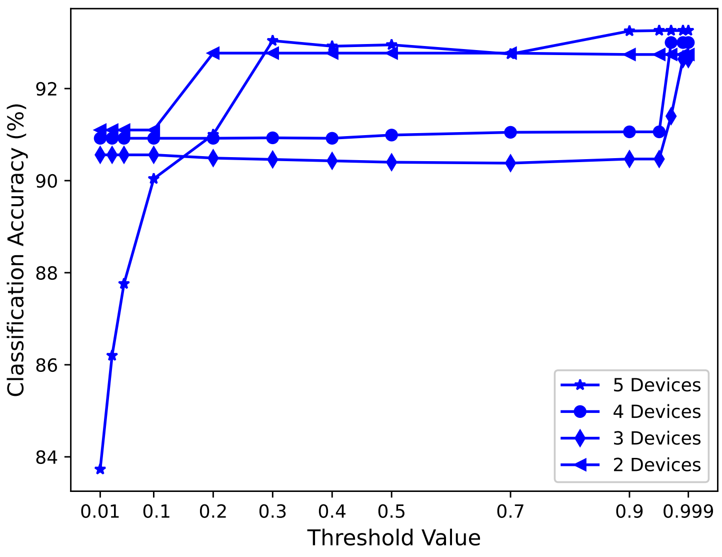


Fig. 9. Impacts of the number of participating devices.

be achieved. As a result, we use two switches as a default value for other experiments. In addition, the highest performance belongs to the case of five devices while the lowest accuracy is with three devices. In terms of threshold value, performance of the proposed architecture can be improved significantly when using a higher threshold value. This is because we tend to wait for the last switch to classify traffic when λ is high.

5. Conclusion

In this work, to address the lack of resources of edge devices, we design a collaborative detection model that consists of multiple sub-models. Each sub-model is executed on a programmable switch and only necessary connections are stored to detect one or several types of traffic. Using experiments with various network parameters, the designed model is proved to be more lightweight than the multi-label classifier without a reduction in classification performance, which allows more traffic to be examined at a switch. Moreover, we find a trade-off between classification accuracy and model complexity when choosing network parameters such that a threshold value, pruning rate, and the number of participating devices. For future work, we plan to design an assignment algorithm that assigns traffic classes to switches under different constraints such as remaining resources and traffic speed. Moreover, we will address the isolation problem in which at least one output neuron is isolated from the pruned network.

Acknowledgment

This research is funded by Vietnam National Foundation for Science and Technology Development (NAFOSTED) under grant number 102.02-2020.06

References

[1] T.-N. Dao and H. J. Lee, "Stacked autoencoder-based probabilistic feature extraction for on-device network intrusion detection," *IEEE Internet of Things Journal*, 2021. doi: 10.1109/JIOT.2021.3078292

[2] Z. Ahmad, A. Shahid Khan, C. Wai Shiang, J. Abdullah, and F. Ahmad, "Network intrusion detection system: A systematic study of machine learning and deep learning approaches," *Transactions on Emerging Telecommunications Technologies*, vol. 32, no. 1, p. e4150, 2021. doi: https://doi.org/10.1002/ett.4150

[3] G. Andresini, A. Appice, and D. Malerba, "Autoencoder-based deep metric learning for network intrusion detection," *Information Sciences*, vol. 569, pp. 706–727, 2021. doi: https://doi.org/10.1016/j.ins.2021.05.016

[4] F. Erlacher and F. Dressler, "On high-speed flow-based intrusion detection using snort-compatible signatures," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–1, 2020.

[5] M. F. Umer, M. Sher, and Y. Bi, "Flow-based intrusion detection: Techniques and challenges," *Computers And Security*, vol. 70, pp. 238–254, 2017. doi: https://doi.org/10.1016/j.cose.2017.05.009

[6] F. Erlacher and F. Dressler, "Fixids: A high-speed signature-based flow intrusion detection system," in *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, 2018. doi: 10.1109/NOMS.2018.8406247 pp. 1–8.

[7] G. Wang, Z. Liu, S. Zhuang, B. Hsieh, J. Gonzalez, and I. Stoica, "Sensai: Fast convnets serving on live data via class parallelism," in *MLOps Systems workshop in MLSys*, 2020.

[8] K. Hyunjae, A. Dong Hyun, L. Gyung Min, Y. Jeong Do, P. Kyung Ho, and K. Huy Kang, "Iot network intrusion dataset," 2019. doi: 10.21227/q70p-q449

[9] F. Paolucci, F. Civerchia, A. Sgambelluri, A. Giorgetti, F. Cugini, and P. Castoldi, "P4 edge node enabling stateful traffic engineering and cyber security," *Journal of Optical Communications and Networking*, vol. 11, no. 1, pp. A84–A95, 2019.

[10] A. Agrawal and C. Kim, "Intel tofino2–a 12.9 tbps p4-programmable ethernet switch," in *2020 IEEE Hot Chips 32 Symposium (HCS)*. IEEE Computer Society, 2020. doi: 10.1109/HCS49909.2020.9220636 pp. 1–32.

[11] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[12] M. Budiu and C. Dodd, "The p416 programming language," *ACM SIGOPS Operating Systems Review*, vol. 51, no. 1, pp. 5–14, 2017.

[13] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: An ensemble of autoencoders for online network intrusion detection," in *Network and Distributed Systems Security (NDSS) Symposium 2018*, 2018. doi: https://doi.org/10.48550/arXiv.1802.09089

[14] T.-N. Dao, V.-P. Hoang, C. H. Ta *et al.*, "Development of lightweight and accurate intrusion detection on programmable data plane," in *2021 International Conference on Advanced Technologies for Communications (ATC).* IEEE, 2021. doi: 10.1109/ATC52653.2021.9598239 pp. 99–103.

[15] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[16] "Pruning by explaining: A novel criterion for deep neural network pruning," *Pattern Recognition*, vol. 115, p. 107899, 2021. doi: https://doi.org/10.1016/j.patcog.2021.107899

Manuscript received 02-08-2022; Accepted 09-11-2022

■

Thi Nga Dao received a B.S. degree in Electrical and Communication Engineering from Le Quy Don Technical University, Vietnam in 2013, an M.S. degree in Computer Engineering from University of Ulsan in 2016, and a Ph.D. degree in Computer Engineering from University of Ulsan, South Korea in 2019. She was a postdoctoral fellow at Intelligent Networked Systems Lab (INSLab) of Ewha Womans University from 2020 to 2021. From July 2019, she has been working as a lecturer in Faculty of Radio-Electronic Engineering, Le Quy Don Technical University, Hanoi, Vietnam. Her research interests include machine learning-based applications in network security, network intrusion detection and prevention systems, human mobility prediction and mobile crowdsensing.
Email: daothinga@mta.edu.vn

Manh Hung Tran received the B.S. degree in Electrical and Communication Engineering and the M.S. degree in Electrical Engineering from Le Quy Don Technical University, Vietnam in 1998 and 2003, respectively. His current research interests include Machine Learning, Network Security.
Email: trmhung@gmail.com

Van Duc Le is a research fellow at School of Computer and Engineering, Nanyang Technological University, Singapore. Previously, he was a research fellow (2016-2018) at Department of Computer Science, National University of Singapore. He received the BEng degree in electronics and telecommunications engineering from Le Quy Don Technical University, Vietnam, in 2011 and the PhD degree in computer engineering from University of Ulsan, South Korea, in 2016. His research interests include sensor networks, IoT networked sensing and computing in cyber-physical systems. He is a Senior Member of IEEE.
Email: vdle@ntu.edu.sg

HỆ THỐNG PHÁT HIỆN XÂM NHẬP MẠNG HỢP TÁC CHO LỚP DỮ LIỆU LẬP TRÌNH ĐƯỢC

Đào Thị Ngà, Trần Mạnh Hùng, Lê Văn Đức

Tóm tắt

Để phát hiện và phản ứng sớm với những mối nguy hại trong mạng, hệ thống phát hiện xâm nhập mạng nên được hiện thực ở trên lớp dữ liệu. Tuy nhiên, mô hình phát hiện xâm nhập mạng dựa trên kỹ thuật học máy tiên tiến có độ phức tạp lớn trở nên không phù hợp với các chuyển mạch có tài nguyên hạn chế. Để giải quyết vấn đề này, chúng tôi đề xuất một mô hình phát hiện hợp tác gọn nhẹ lấy cảm hứng từ ý tưởng phân loại song song và cắt giảm nơ-ron. Cụ thể, mô hình phân loại đa nhãn truyền thống được chia thành một số mô hình con dành riêng cho từng nhãn dữ liệu và mỗi mô hình con phụ trách phát hiện một hoặc một số lớp dữ liệu. Trong mô hình của chúng tôi, số lượng thiết bị chuyển mạch tham gia có thể thay đổi dựa trên lưu lượng mạng và lượng tài nguyên sẵn có của các thiết bị mạng. Hơn nữa, để giảm kích thước của các mô hình con, phương pháp cắt giảm nơ-ron dựa trên trọng số được áp dụng cho từng mô hình con nhằm giữ lại các kết nối quan trọng. Mô hình được đánh giá với các tham số mạng khác nhau và kết quả cho thấy kiến trúc đề xuất có độ phức tạp thấp hơn nhiều so với bộ phân loại đa nhãn truyền thống mà không làm giảm chất lượng phân loại tấn công mạng.

Từ khóa

Quản lý lưu lượng mạng, phát hiện xâm nhập mạng, cắt giảm nơ-ron