JointNIDS: Efficient Joint Traffic Management for On-Device Network Intrusion Detection

Thi-Nga Dao and HyungJune Lee, *Member, IEEE*

***Abstract*—Data plane programmability enables the embedding of a network intrusion detection system (NIDS) on programmable switches to dynamically control the efficiency of attack type detection and the overhead in the computation and network side. However, it is a challenging task to implement a feasible embedded detection model with advanced machine learning techniques such as deep learning. It is due to the limited support provided by programming languages on the data plane and the computing resource constraints at the edge. We propose a joint traffic classification architecture called *JointNIDS* that splits a classification model into two sequential sub-models. In this model, the primary switch is dedicated to major attack classification. The secondary switch is used mainly for a further in-depth inspection of the rest of the minor traffic types. The presence of some partially overlapping hidden units in the two sequential switches can help to reduce the computational overhead at the edge, while increasing the packet inspection throughput. Experimental results on the P4 framework demonstrate that *JointNIDS* has reduced attack detection time, while achieving a similar accuracy performance, as other counterpart algorithms. To further develop the proposed architecture, *JointNIDS* implements an optimization step. It maximizes the amount of data to be inspected by a system, taking into account the constraints of computing resources and network bandwidth for a given performance requirement. We validate the effectiveness of collaborative joint optimization in various scenarios.**

***Index Terms*—Anomaly Classification, joint detection, On-Device AI, network intrusion detection system.**

I. Introduction

A. Aim and Motivation

DUE to recent advances in Internet of Things (IoT) technologies, the IoT is playing an important role in various emerging applications, such as smart cities, agriculture, manufacturing, and healthcare. Due to the lack of security methods in IoT, there has been a tremendous increase in both cyber-attack volume and data breaches. Therefore, to ensure the security and reliability of the IoT, it is important to quickly detect and classify network threats. The problem of maximizing the amount of data to be inspected and classified in a given IoT network is both challenging and important.

Software-defined networks (SDN) separate the control plane from the data plane and provide a fully programmable and dynamic network architecture that can be changed if needed [26]. Compared to traditional fixed-function switches, data plane programmability allows operators to design customized packet processing functions while reducing the time required to implement new functions or to modify the current protocol. Due to the high flexibility and the reduction in detection time of this approach, the implementation of the security method in the data plane has attracted a lot of attention from both the researchers and practitioners [19], [22], [23], [31].

When deploying the security model in the data plane, there are several some main challenges. First, since programmable switches have equipped with limited computing resource, the traffic classification architecture should also be lightweight. Second, the programming language (e.g., P4) used for the data plane focuses on the packet forwarding function, thus suffering from a lack of the mathematical operations needed for a machine learning-based model. More specifically, only few operations for integer and binary numbers are supported by the P4 language.

Therefore, to ensure the quick detection of network attacks, we need a lightweight and accurate intrusion detection model that can be fully implemented in the data plane. For the traffic classification problem, the well-known traffic datasets [15], [21], [27] show high imbalance between traffic attacks (i.e., there are majority and minority classes). The number of packets in each attack category is used to distinguish among attack types. Attacks with a higher number of packets are called major groups, while the remaining attacks belong to minority groups. Motivated by this fact, in the proposed classification architecture, some dominant traffic types are classified on the primary switch by a specific sub-model that has much lower complexity than the whole model. For minority classes, one of the remaining sub-models is executed on a secondary switch. The reduction of both model complexity and classification time in a balanced way becomes more critical.

B. Related Work

With the aim of developing a lightweight security method, a network intrusion detection system (NIDS) that couples a stacked autoencoder with a network simplification technique

Manuscript received 23 March 2022; revised 20 June 2022 and 21 July 2022; accepted 31 July 2022. Date of publication 11 August 2022; date of current version 19 December 2022. This work was supported in part by the National Research Foundation of Korea(NRF) funded by the Korea government(MSIT) under Grant NRF-2021R1A2B5B01002906 and in part by the Vietnam National Foundation for Science and Technology Development (NAFOSTED) under Grant 102.02-2020.06. The review of this article was coordinated by Dr. Mostafa Fouda. *(Corresponding author: HyungJune Lee.)*

Thi-Nga Dao is with the Faculty of Radio-Electronic Engineering, Le Quy Don Technical University, Hanoi 10000, Vietnam (e-mail: daothinga.mta@gmail.com).

HyungJune Lee is with the Department of Computer Science and Engineering, Ewha Womans University, Seoul 03760, Republic of Korea (e-mail: hyungjune.lee@ewha.ac.kr).

Digital Object Identifier 10.1109/TVT.2022.3198266

was proposed in [5]. However, the implementation of NIDS on the programmable data plane has not been considered and is still an open issue to be addressed.

Most previous studies into network intrusion detection on the data plane usually require an external traffic flow collector (e.g., Cisco NetFlow or Huawei NetStream) connecting to the switch to first extract the network information [8], [9], [13], [33]. Then, the extracted features are used as the inputs to an intrusion detection model that is based on a machine learning technique such as support vector machine, random forest, or neural networks) [2], [28], [34]. The use of the flow collector outside the switch enables the use of the computing resource of the external device. However, this approach also leads to high detection latency due to the transmission delay from the switch to the external collector.

To address the problem of the high detection delay incurred by using the traffic flow collector, there are some studies into the embedding of an intrusion detection function on the programmable data plane [6], [11], [29], [30], [36]. Several studies focus on the detection of heavy hitters, which are defined as network flows with extremely high volumes of traffic. Heavy hitter detection has received considerable attention, since this is a fundamental problem in many applications, including the identification of denial-of-service (DoS) attacks and anomalies.

Some rule-based network intrusion detection models are embedded in the data plane, in which a white-list of secure IP addresses is implemented as a look-up table in the switch [19], [23]. For example, in [23], when a packet has a matching entry in the match-action table, this packet goes through the switch. Otherwise, the second level of examination that links to the SDN controller is considered.

There are some research into the detection of a specific type of attack, such as DoS, based on the estimation of the entropy of the source and destination IP addresses extracted from the incoming packets [3], [7], [24], [32]. These methods seem to be simple and feasible for the programmable data plane. However, intrusion detection for other attack types using entropy-based estimation needs further investigation since information about IP address frequency may not be a good feature for identifying other network attacks like man-in-the-middle or botnet.

Although a number of timely intrusion detection models on the programmable data plane were introduced, a model that considers the detection and classification of multiple network attacks is needed. Advanced machine learning-based classifiers such as neural networks with high classification performance have not been considered in the data plane. It is due to the limited support of programming languages for the data plane. We aim to address these challenges by developing a lightweight and accurate joint classification model on the data plane that can detect and classify the data traffic with low detection latency.

By executing the same arithmetic operations for incoming traffic, the existing detection models have treated dominant and minority traffic labels equally. In contrast, this work takes traffic label frequency into account. We design a collaborative joint classification model for NIDS called *JointNIDS*, which requires fewer operations for major traffic labels at a switch.

C. Contributions

In this work, we raise a key question: "Is there any way to cooperatively classify network traffic using sequential switches with resource constraints in an IoT environments?" To answer the question, we propose a joint traffic classification architecture, *JointNIDS* to effectively reduce model complexity and the latency of intrusion detection. The main idea behind the proposed architecture is that we decompose a classification model into sub-models that can be executed on different switches. The whole network consists of overlapping and non-overlapping parts. The overlapping neurons are shared among switches and the non-overlapping neurons are used only at a specific switch. The collaborative learning architecture includes a primary switch and one or more secondary switches. When a packet arrives at the primary device, this switch computes the overlapping part and its non-overlapping parts to detect some main traffic attacks. If the traffic is predicted to be one of these types, the classification of the packet ends and there is no need to use secondary switches. Otherwise, one of the secondary switches is used to detect whether the packet belongs to a different traffic type. The procedure is repeated until the traffic is classified into a specific type. Due to the high imbalance of data traffic, only the primary switch is required for traffic classification most of the time. Therefore, the number of floating-point operations (FLOPs) and the model complexity can be mitigated to a considerable extent.

To evaluate the joint classification model, we first compare the proposed architecture *JointNIDS* to existing models in terms of classification accuracy and processing delay in the programmable data plane. Then, we develop an optimization problem to maximize the amount of data to be classified under different constraints on computing resources, communication cost, and classification accuracy. We assume that the number of packets arriving at each switch follows a uniform distribution. The completion ratio of the number of classified packets to the number of generated packets is used as a performance metric to evaluate the effects of the proposed joint classification model. The salient contributions of our work are summarized as follows.

- We propose a collaborative learning architecture, *JointNIDS*, for traffic classification that can be trained and executed on programmable switches. It allows us to quickly detect and classify network threats by reducing the model complexity.
- Using a programmable data plane simulator, we determine the optimal parameters for the proposed model, including the overlapping ratio and network architecture. The classification accuracy and latency of our model are compared with existing methods to show the effectiveness of joint classification.
- An optimization problem is formulated to find the maximum number of packets to be managed, in two cases: with and without joint classification models. The experimental results show that using our proposed model produces more effective traffic management than previous approaches. *JointNIDS* meets the requirements for performance and computing resources in a home IoT environment.

TABLE I
COMPARISONS OF RELATED WORK WITH OUR WORK

References	Main Features
[8], [9], [13], [33]	Require an external traffic collector to inspect data, no on-device intrusion detection
[6], [11], [29], [30]	Heavy hitter detection, on-device detection
[19], [23]	Rule-based intrusion detection, on-device detection
[7], [24]	DDoS attack detection based on entropy estimation, on-device detection
JointNIDS	Lightweight NN-based on-device intrusion detection and classification in a series of switches

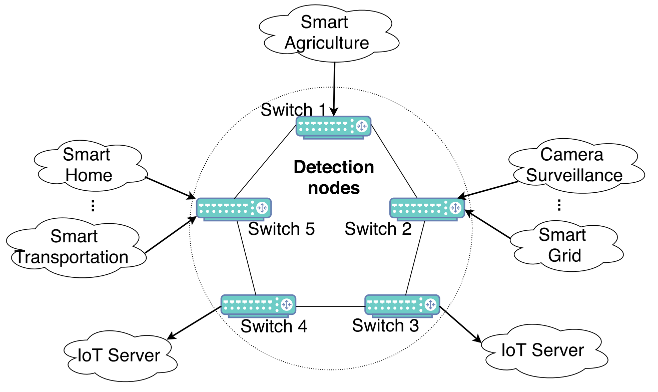

Fig. 1. Intrusion detection system in the IoT network. Switches take in charge of detecting abnormal traffic in the network.

II. SYSTEM MODEL

We consider heterogeneous IoT networks that connect IoT devices for various applications such as smart homes, smart agricultural systems or smart transportation systems as shown in Fig. 1. Data packets generated by IoT devices are transmitted to one or more IoT servers via multiple network switches in series for environment monitoring and decision making. The switches are connected using a bus, star, tree, ring, or mesh network topology. As an example, a ring network topology consisting of five switches is shown in Fig. 1.

As these switches are usually programmable, allowing them to be fully customized to meet user needs, data packets can be processed by the data plane, using, for example, the P4 programming language [4]. P4-supported switches can execute data forwarding as well as network intrusion detection for incoming packets. We assume that an IoT network consists of S switches that classify data traffic into different types: normal, denial of service, man-in-the-middle, reconnaissance, botnet, or other. Let N_i denote the number of packets arriving at switch i per second ($i = \{1, 2, \ldots, S\}$ where S is the total number of switches in the network topology). These packets belong to one of the above-mentioned traffic types.

After arriving at a switch, incoming data packets are passed through the packet header parser to extract their statistical features, such as arrival rate and time elapsed from the last appearance). These features are then fed into a detection model to compute the probability of a network threat from the received data packets. If the probability is greater than a certain threshold value, then the defense system may take an action, such as producing an alarm warning or packet drop, to prevent some possible damages that may be caused by the detected threats. Otherwise, data packets are considered to be normal, and should be forwarded to a designated IoT server.

In this work, we aim to develop a system to quickly detect and classify a possible intrusion, to facilitate early responses to intrusion behaviors. It requires an efficient traffic classification model to be implemented in the data plane. However, due to the limited computing resources on the switch side, and the rapidly increasing amount of traffic in the IoT environment, there is a clear need for a lightweight NIDS architecture. It helps to well adapt to the dynamically changing resource constraints and traffic flows.

We propose a lightweight and joint classification architecture that is suitable for networking devices with limited resources in Section III, and an optimized traffic management strategy in Section IV.

III. JOINT CLASSIFICATION OF TRAFFIC IN THE DATA PLANE

Neural networks (NNs) are machine learning models that can accurately learn non-linear mappings from input features to output values. Due to the high complexity of the architecture of most NNs, it is challenging to implement an NN-based model in the data plane at the limited-resource edge devices.

To effectively leverage a limited computational resource over multiple switches, we decompose the whole NN-based model into sub-models that can be trained and executed into multiple switches. Model decomposition or parallelism has been described in the literature [10], [14], [16], [35]. However, the existing model decomposition approaches distribute the computing load onto multiple devices, without reducing the total number of FLOPs. In the IoT environments, it is necessary to reduce some redundant computation as far as possible, for example by extracting some common computational parts from the architecture.

We propose a unique model decomposition strategy that allows fewer parameters to be exchanged between devices. It transmits a subset of parameters relevant to several effective neurons that can be shared by devices for training. These shared neurons are called *overlapping* neurons.

The whole network is classified into two parts, overlapping and non-overlapping. The overlapping parts are shared between devices, while the non-overlapping parts are needed at each specific device. There is usually an imbalance in the distribution of the data between the dominant and non-dominant classes.

We divide the NN-based model in such a way that the most frequent traffic classes are classified by a specific sub-model that computes the overlapping units, and its own non-overlapping units. If the traffic does not belong to one of the most dominant

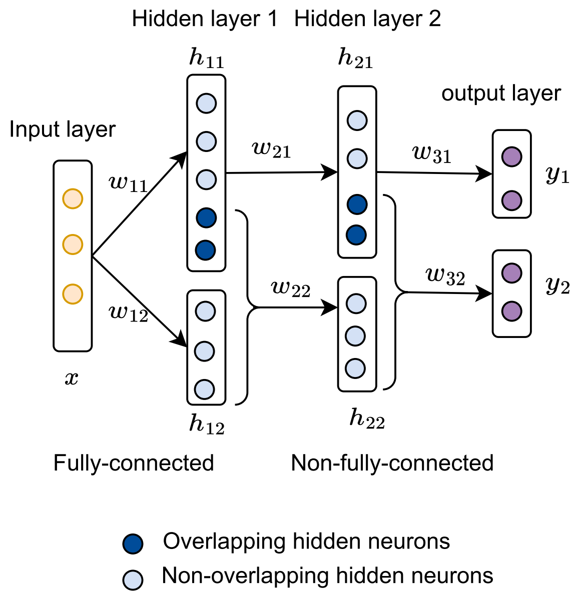

Fig. 2. The proposed joint traffic classification architecture *JointNIDS*. Primary and secondary switches compute the probability of dominant (y_1) and minority attacks (y_2), respectively.

classes, another sub-model is needed at the next device. In the following sub-model, its own non-overlapping units need to be computed together with the overlapping parts inherited from the prior sub-model. This step is repeated until the traffic is classified into a specific class. In summary, the whole network is vertically divided into a series of overlapping sub-models vertically. Each sub-model is in charge of estimating the probability of traffic classes.

We present a collaborative learning architecture of the NN-based joint classification model with two sub-models, as illustrated in Fig. 2. The features extracted from the packet header are fed into input units, x. The NN model is used to learn a non-linear mapping from the inputs x to the outputs y, together with the probability of attack types. It is supposed that the probability of attack types depends upon a portion of the hidden units in the previous layers. The first sub-model aims to predict the attack probability of m_1 classes, while the remaining $(m - m_1)$ outputs are estimated by its following sub-model.

The joint traffic classification model learns the non-linear mapping from the packet features, x, to the probability values of traffic classes, y. We compute the activation function g using the following relations:

$$h_{11} = g(x, w_{11}, b_{11}), \quad (1)$$

$$h_{21} = g(h_{11}, w_{21}, b_{21}), \quad (2)$$

$$y_1 = \sigma(h_{21}, w_{31}, b_{31}), \quad (3)$$

$$h_{12} = g(x, w_{12}, b_{12}), \quad (4)$$

$$h_{22} = g\left(h_{11}[n_{no,1} :] \cup h_{12}, w_{22}, b_{22}\right), \quad (5)$$

$$y_2 = \sigma(h_{21}[n_{no,2} :] \cup h_{22}, w_{32}, b_{32}), \quad (6)$$

$$y = y_1 \cup y_2, \quad (7)$$

where w_{11}, w_{21}, b_{11}, and b_{21} denote the weights and biases of the first sub-model, while w_{12}, w_{22}, b_{12}, and b_{22} are those for the second sub-model. Eqs. (1) to (3) apply to the sub-model 1, and the (4) to (6) are for the sub-model 2. The number of non-overlapping units in the first and second hidden layers are denoted by $n_{no,1}$ and $n_{no,2}$, respectively. Let r_o denote the ratio of the overlapping units in each hidden layer ($0 \leq r_o \leq 1$), and $n_{h,i}$ denotes the number of hidden units in layer i. Then, the number of non-overlapping units $n_{no,i}$ in the hidden layer i of the first sub-model can be calculated as:

$$n_{no,i} = \left\lceil \frac{n_{h,i}(1 - r_o)}{2} \right\rceil. \quad (8)$$

We implement the joint traffic classification model in the P4 framework [4]. Since the P4 language does not support non-linear operations, the linear ReLU function is selected as the activation function for the hidden layers. The sigmoid function (σ) is used to compute the outputs y_1 and y_2 of the first and second sub-models. The output units (y) are a concatenation of y_1 and y_2, which are the outputs of sub-models 1 and 2, respectively.

The joint traffic classification model is trained over two or more switches in a sequential manner. With two switches, the sub-model 1 is trained first at the primary switch to find the optimal parameters w_{11}, w_{21}, b_{11}, and b_{21} by minimizing the mean squared error (MSE)-based loss function L_1. We define N as the number of training samples, while $y_1^{(j)}$ and $\hat{y}_1^{(j)}$ are the true and the predicted outputs of sub-model 1, respectively. The loss function L_1 is computed as:

$$L_1 = \frac{1}{N}\sum_{j=1}^{N}(y_1^{(j)} - \hat{y}_1^{(j)})^2. \quad (9)$$

Then, the parameters of the sub-model 2, including w_{12}, w_{22}, b_{12}, and b_{22} are trained at the secondary switch. We use the MSE-based cost function given the pre-trained parameters of the first sub-model. The loss function L_2 of the sub-model 2 is calculated as below.

$$L_2 = \frac{1}{N}\sum_{j=1}^{N}(y_2^{(j)} - \hat{y}_2^{(j)})^2, \quad (10)$$

where $y_2^{(j)}$ and $\hat{y}_2^{(j)}$ are the true and the predicted outputs of sub-model 2.

For inference, the primary switch first extracts the features of an incoming packet and sends them to the input layer of the sub-model 1. Then, the sub-model 1 calculates the probability of m_1 major traffic labels, as shown in Fig. 3. If the maximum probability of $\hat{y}_1$ exceeds a threshold value λ (e.g., $\lambda = 0.5$), it is highly likely that the incoming traffic belongs to one of m_1 major classes. Therefore, the label of the packet can be predicted using only the sub-model 1. Otherwise, if the major class prediction turns out to be not so definitive, i.e., $\max(\hat{y}_1) < \lambda$, there may not be strong evidence with which to deduce the traffic label from the perspective of the primary switch. Then, the secondary switch becomes engaged to execute the sub-model 2 to obtain $\hat{y}_2$. Combined with the output of the first sub-model $\hat{y}_1$, the packet is finally classified into label $l = \text{argmax}\, \hat{y}$. A flowchart

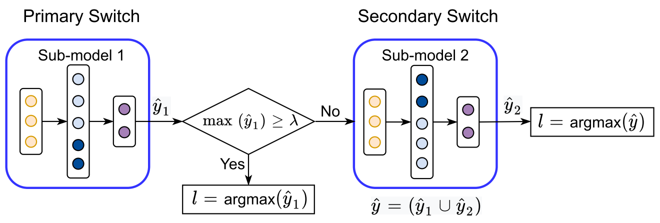

Fig. 3. The inference phase of the proposed model. When major attacks are injected to the network, only the sub-model 1 is executed at the primary switch, whereas the additional sub-model 2 becomes involved for a further examination of minor attacks at the secondary switch.

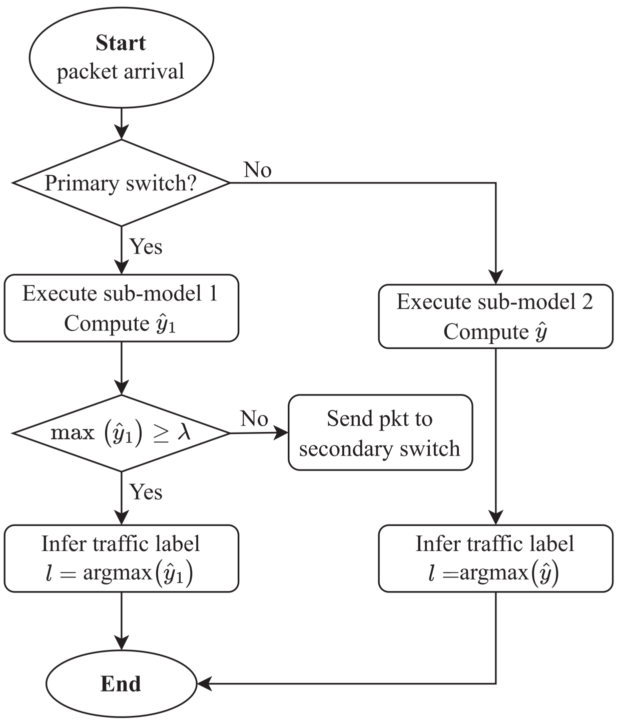

Fig. 4. Flowchart of *JointNIDS*. In the case of correct prediction, dominant and minority attacks are detected at the primary and the secondary switches, respectively.

of the proposed architecture is shown in Fig. 4. When a packet arrives at the primary switch, the output $\hat{y}_1$ from sub-model 1 is computed. If the packet is predicted to be part of a the major attack, the traffic classification procedure is ended. Otherwise, the incoming packet is transmitted to the secondary device, the sequential sub-model 2 is executed, and its traffic class is further examined.

IV. Maximization of Traffic Management

In the IoT environment, in which resource-constrained devices mostly perform minimally effective jobs, it is important for the on-device network intrusion detection function to be resilient and flexible at the architecture level. Also, the detection function needs to reduce the detection demands from the possible intermittent traffic surge.

TABLE II
List of Main Notations in ILP Formulation

Switch parameters	
S	Number of switches
$\mathbb{S}$	Set of switches
N	Total number of generated packets per second
N_i	#Generated packets sent to switch i per second
n_{ij1}	#packets classified by switch i and 1st part of model j
n_{ij2}	#Packets classified by switch i and 2nd part of model j
β_{ij1}	Maximum value of n_{ij1}
β_{ij2}	Maximum value of n_{ij2}
n	#Packets classified by all switches
γ_i	Maximum number of offloading packets towards switch i
$t_{c,i}$	Classification time constraint for switch i
Classification model parameters	
C	Number of classification models
$\mathbb{C}_1$	Set of classification models executed on 1 switch
$\mathbb{C}_2$	Set of classification models executed on 2 switches
A_j	Classification accuracy of model j
T_{j1}	Average classification time using first part of model j
T_{j2}	Average classification time using second part of model j
a_{req}	Classification accuracy requirement

To more actively respond to the traffic dynamics and increase the traffic detection capability, we take an optimization approach by formulating the joint classification problem as an integer linear program (ILP). The main objective is to maximize the number of packets that can be inspected under the constraints of classification accuracy, communication cost, and computing resources. Table II presents the main symbols used in the ILP formulation.

The system consists of a set of S switches denoted by $\mathbb{S} = \{1, 2, \ldots, S\}$, in which switch i receives N_i packets per second from an IoT network. Then, the total number of generated packets per second is $N = \sum_{i=1}^{S} N_i$. It is assumed that there are C available classification models, which are divided into two sets: $\mathbb{C}_1$ and $\mathbb{C}_2$ are the sets of models to be executed with one switch and two switches in series, respectively. Let T_{j1} and T_{j2} denote the average execution time for classifying a single packet using model j on the primary and secondary switches, respectively. The average execution time is measured on the P4-supported switches in the data plane. If model j belongs to $\mathbb{C}_1$, traffic classification is executed on the primary switch, without the involvement of the secondary switch ($T_{j2} = 0$).

We define n_{ij1} and n_{ij2} as the assignment variables of the number of packets classified by the primary switch with model i and the secondary switch with model j. Note that $n_{ij2} = 0$ for $j \in \mathbb{C}_1$, since the models in $\mathbb{C}_1$ are executed only at the primary switch. In terms of classification time, the average classification time for switch i to finish classifying all the assigned packets should be $\sum_{j=1}^{C} (T_{j1} n_{ij1} + T_{j2} n_{ij2})$. The execution time at switch i for traffic classification should not exceed the maximum classification time, $t_{c,i}$. For example, $t_{c,i} = 1$ is the allowed deadline of 1 s, in the case that there is no other tasks required by the switches. We empirically measure the classification time for various models as T_{j1} and T_{j2}. Then, the maximum limits of the assignment variables β_{ij1} and β_{ij2} can be derived as below:

$$\beta_{ij1} = \frac{1}{T_{ij1}}, \tag{11}$$

$$\beta_{ij2} = \frac{1}{T_{ij2}}. \tag{12}$$

In order to reduce the communication cost, each switch is encouraged to classify as many as possible its incoming packets. Since the number of the incoming packets can be different from the number of packets that can be processed at the switch itself, we introduce a constraint, γ_i, such that the difference should be less than γ_i, i.e., $\sum_{j=1}^{C} n_{ij1} - N_i \leq \gamma_i, \forall i$. In the ideal case, $\gamma_i = 0$ means switch i is able to classify all the incoming packets, without offloading some of them to the neighboring switches and without paying communication costs for traffic classification.

The ILP can be formulated as follows:

$$\max \quad \sum_{i=1}^{S}\sum_{j=1}^{C} n_{ij1}, \tag{13}$$

subject to

$$\frac{1}{n}\sum_{i=1}^{S}\sum_{j=1}^{C} A_j n_{ij1} \geq a_{req}, \tag{14}$$

$$\sum_{i=1}^{S}\sum_{j=1}^{C} n_{ij1} = n, \tag{15}$$

$$\sum_{j=1}^{C}(T_{j1}n_{ij1} + T_{j2}n_{ij2}) \leq t_{c,i}, \quad \forall i \in \mathbb{S}, \tag{16}$$

$$t_{c,i} \leq 1, \quad \forall i \in \mathbb{S}, \tag{17}$$

$$\sum_{i=1}^{S} n_{ij1} = \sum_{i=1}^{S} n_{ij2}, \quad \forall j \in \mathbb{C}_2, \tag{18}$$

$$\sum_{j=1}^{C} n_{ij1} - N_i \leq \gamma_i, \quad \forall i \in \mathbb{S}, \tag{19}$$

$$\beta_{ij2} = 0, \quad \forall j \in \mathbb{C}_1, \tag{20}$$

$$0 \leq n_{ij1} \leq \beta_{ij1}, \quad \forall i \in \mathbb{S}, \forall j \in \mathbb{C}, \tag{21}$$

$$0 \leq n_{ij2} \leq \beta_{ij2}, \quad \forall i \in \mathbb{S}, \forall j \in \mathbb{C}. \tag{22}$$

The objective function in (13) maximizes the total number of packets classified by all of the participating switches in the network. The constraints in (14) and (15) make sure that the average classification accuracy over all switches and models should achieve at least the requirement value, a_{req}. The constraints in (16) and (17) restrict the execution time for traffic classification at each switch to be less than the maximum value, $t_{c,i}$. (18) ensures that the traffic classification models in $\mathbb{C}_2$ are executed over two switches. Then, the communication overhead should not go beyond a given constraint in case that a switch sends the packet features to the other switches for classification, as in (19). The remaining constraints in (20), (21), and (22) present the range of the assignment integer variables n_{ij1} and n_{ij2}.

If there is one classification model, $C = 1$, that belongs to ($\mathbb{C}_1$), the ILP formulation is similar to the bounded knapsack problem [25], which is an NP-hard problem. Switches are considered as bags with limited capacity, while the packets are the items to be assigned to switches. There are multiple classification models, and a suitable model needs to be selected for a packet to be classified with a single switch. The formulated combinatorial optimization problem turns out to be NP-hard, which is actually more difficult than the knapsack problem. Although the optimal solution cannot be found in the polynomial time, there are several heuristic algorithms [18] that can be used to approximate the solution for our optimization problem case. As a proof-of-concept prototype, a simple greedy algorithm is applied to solve the optimization problem. For each incoming packet, a group of switches select a classification model with the lowest detection time so that both accuracy and time constraints can be satisfied. The assignment is completed when at least one constraint is violated, or all packets are assigned.

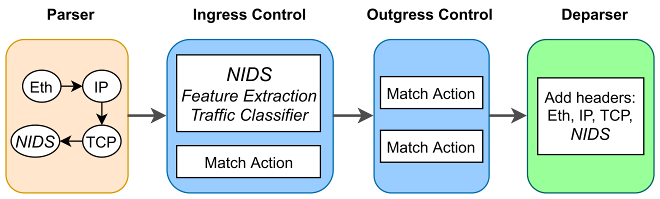

Fig. 5. NIDS with feature extractor and traffic classifier at a P4-supported switch. The input features of *JointNIDS* derived from packet headers are fed into the traffic classifier to compute the probability of attack types.

V. Experimental Results

A. Implementation of NIDS in the Data Plane

To validate our approach, we implemented the joint classification architecture, *JointNIDS* in the P4 framework. The main elements of P4-supported switch architecture, including parser, ingress control, outgress control, and deparser, are shown in Fig. 5. In the ingress and outgress blocks, there is a set of *Match Action* stages in which data packets are matched against a table that consists of some entries to execute the appropriate action. The intrusion detection system implemented at the ingress control block consists of two parts: feature extraction and traffic classification. For the first part, stateful objects, or more precisely programmable registers, are used to store and update packet features. These registers are initialized to 0. Whenever a packet arrives at a switch, necessary information such as source/destination IP address, packet length, and arrival time are extracted from the packet header by the parser. Due to the limited computing resources available in the data plane, we selected a set of six features from [20] as the input of the classification model. The previous features taken from the registers are used to compute the current packet's features. Then, the current features are written to registers at the index specified by the packet information. It is assumed that 32 bits are used to store features using signed integer numbers. To make sure all features have a similar value range, feature normalization is considered by applying the bit shift operation in the P4 programming language. The bit shift operation can be used instead of the integer division operation, which is not supported by P4.

TABLE III
Data Distribution of the IoT Network Intrusion Dataset

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

The normalized features are injected into the classifier to predict the most likely traffic label. In the joint traffic classification model, the outputs of sub-model 1 are computed first at the primary switch. If there is an output value greater than a threshold value, the label can be inferred. Otherwise, the overlapping hidden values and the outputs of the first sub-model are transmitted to the secondary switch. We define a custom header called NIDS that includes the overlapping hidden and output values of sub-model 1. A custom header is added to the packet at the deparser block of the architecture, as shown in Fig. 5. At the secondary switch, the NIDS header is extracted, and then the outputs of sub-model 2 are computed, to predict the traffic label using the outputs of both sub-models.

B. Network Setup

To evaluate the feasibility of joint classification models, we conduct experiments on P4 switches using the network emulator Mininet [17]. There are two hosts, sender and receiver, connected via two serial switches, primary and secondary, that are embedded with the traffic classification function. Data packets generated at the sender follow traffic provided by an IoT network intrusion dataset [15]. There are nearly 3 million packets classified into normal data and one of four network attacks: reconnaissance, man-in-the-middle, denial of service, or botnet as shown in Table III. The top-k traffic types belong to the majority group (where $k = 3$ in our experiments), or the attacks accounting for more than 90% of network threats are called major. For our evaluation, the majority class includes normal, man-in-the-middle, and botnet. The network parameters are trained using 80% of the data from the dataset, while the remaining data are used for performance evaluation. The experiments are conducted in a desktop PC with an Intel Core i7 2.5 GHz CPU with the Radeon R9 M370X 2048 MB and Intel Iris Pro 1536 MB GPU support, and 16 GB RAM, which is comparable to a normal or low-end switch specifications. Two metrics are used to evaluate the classification models, classification accuracy and end-to-end delay from the sender to the receiver.

C. Joint Classification

We first validate the training process of our joint classification model by measuring the loss function as shown in Fig. 6. He initialization [12] is used to initialize network parameters. We use a network with 6 input, 10 hidden, and 5 output units. The batch size is set to 2,048, with eight different learning rates from 0.001 to 0.08 for evaluation. There are two separate steps: in step 1, only the parameters of sub-model 1 are learned, and in step 2, the parameters of sub-model 2 are trained on top of the already-trained and fixed parameters of sub-model 1. The training process at each step stops when there is no further improvement on the validation set for the latest 20 epochs. As shown in Fig. 6, sub-models 1 and 2 took 187 and 253 epochs, respectively, until convergence.

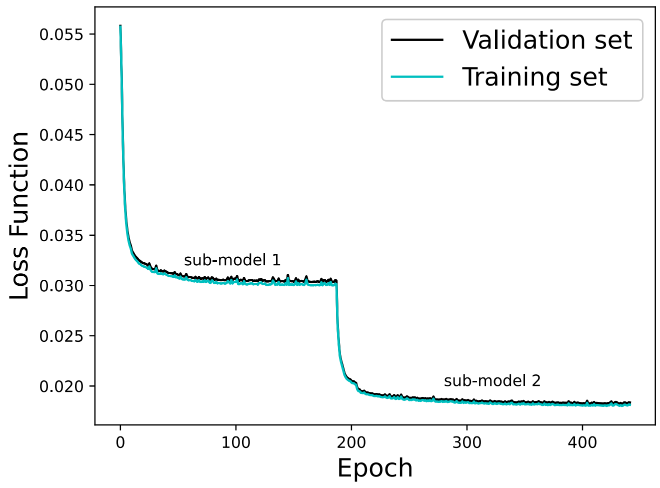

Fig. 6. Loss function with respect to the number of epochs. The parameters of sub-model 1 and 2 are trained by the primary and secondary switches, respectively. The loss value decreases as training continues with epochs.

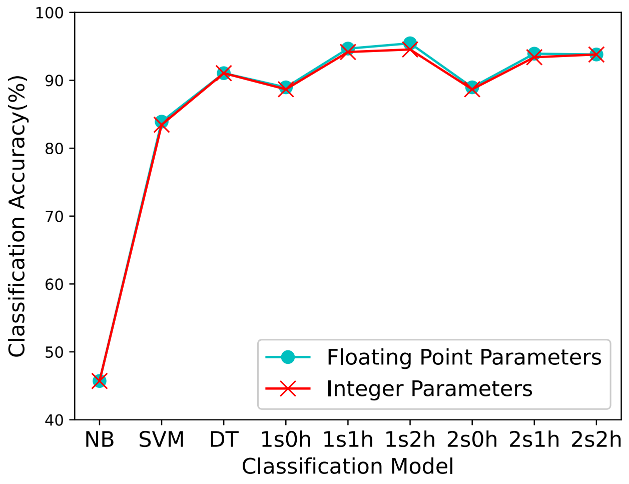

Fig. 7. Comparison of the accuracy of floating-point and integer parameters. *JointNIDS* has a similar performance to that of FC models, and higher accuracy than NB, SVM, and DT.

Then, we investigate the classification accuracy of different traffic classification models: support vector machines (SVM), decision trees (DT) [22], and Naive-Bayes (NB) [23], as shown in Fig. 7. The experiments are conducted using a specific programming language P4, which was designed for programmable switches. To indicate a fully connected (FC) model that is executed with only one switch, the symbol *1 s* is used. Meanwhile, the symbol *2 s* is used for the two switch usage in the joint classification architecture involving with the primary and secondary switches. As the number of hidden layers varies from 0 to 2, the symbols *0 h*, *1 h*, and *2 h* indicate zero, one, and two hidden

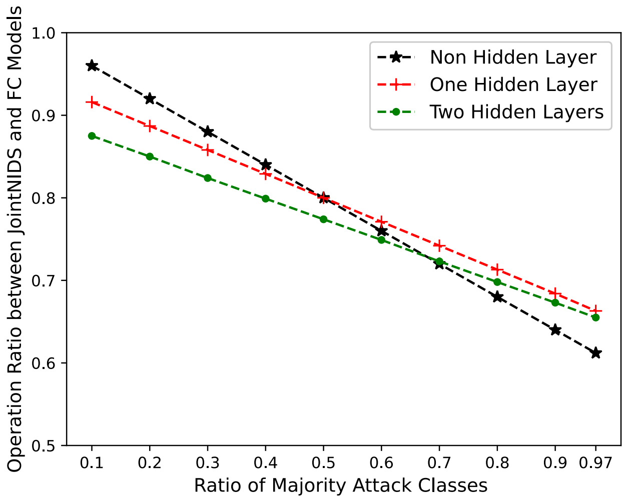

Fig. 8. Comparison of the complexity of the *JointNIDS* and FC models. *JointNIDS* reduces model complexity compared to FC architectures, especially with more imbalanced attack type data.

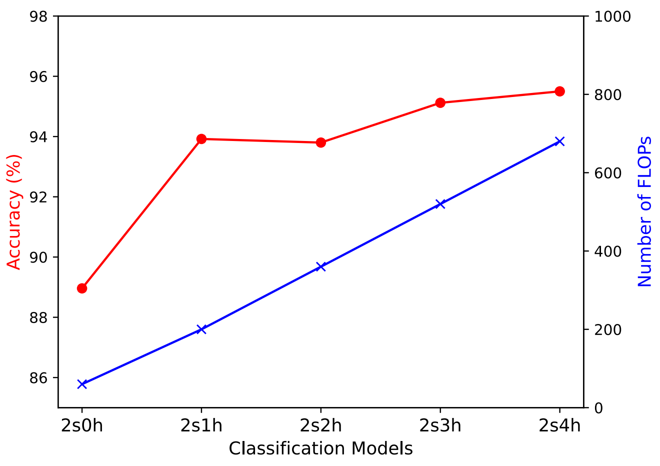

Fig. 9. Impacts of the number of hidden layers in *JointNIDS*. As the number of hidden layers varies from 0 to 4 with the two switch scenario, the accuracy increases, while its resulting computational complexity linearly increases.

layers, respectively. To embed the classification models in the P4-supported switches that do not support floating-point parameters, we round the floating-point parameters learned from the training process to integer values using 10 bits for the fractional numbers. Fig. 7 shows a comparison of the performances of the models using floating-point and integer parameters. Generally, using integer parameters results in slightly lower accuracy than the use of intact floating-point parameters. Our joint classification architecture on the P4-supported switches therefore does not lose fidelity due to integer quantization, compared to the original floating-point parameter-based model on more powerful devices. Table IV compares other performance metrics of the classification models.

The models over two switches (*2s0h*, *2s1h*, and *2s2h*) tend to achieve similar performance to the models on only one switch (*1s0h*, *1s1h*, and *1s2h*). In our joint classification approach, each of two devices are in charge of only its own partial classification model in a distributed sequential manner. Accordingly, it can

TABLE IV
COMPARISON OF CLASSIFICATION MODEL PERFORMANCE

Models	Precision	Recall	F1 Score
NB	0.730	0.457	0.375
SVM	0.810	0.839	0.810
DT	0.882	0.911	0.894
1s0h	0.854	0.887	0.867
1s1h	0.942	0.948	0.940
1s2h	0.943	0.946	0.942
2s0h	0.853	0.885	0.866
2s1h	0.935	0.943	0.929
2s2h	0.950	0.955	0.951

reach its ideal upper bound based on the complete classification model running on a single device. *JointNIDS* allows the training of a classification model over two edge devices by reducing the computational burden, while maintaining similar classification accuracy. For example, the models of *1s2h* and *2s2h* produce 94.54% and 93.80% accuracy, respectively, on the test set on the P4-supported switch; a difference of only 0.74%. On the other hand, the NB-based model has the lowest accuracy of 45.7%. This result is because the NB-based model uses only the source and destination IP addresses as flow signatures for detecting the attack class, even though a single host can generate multiple different attacks. Hence, the classification performance of the NB-based model is lower than the others. Our joint traffic classification models produced better performance than SVM or DT, due to the non-linear mapping from the input features to the output values.

We now examine the performance of *JointNIDS* including classification accuracy and model complexity with the number of hidden layers changes from 0 to 4. The number of floating-point operations (FLOPs) of both sub-models 1 and 2 is used to represent model complexity of *JointNIDS*. As shown in Fig. 9 , the classification accuracy improves as a larger number of hidden layers was used. However, there is a trade-off between classification performance and model complexity. Since our work aims to implement a lightweight detection model on resource-constrained devices, a relatively small neural network is used. Therefore, an NN-based classification model with one hidden layer is used as the default architecture for our experiments.

To show the low complexity of *JointNIDS*, we compare the number of arithmetic operations – multiplication and addition – of the proposed framework and single switch-based FC models with various ratios of the major attack classes (Fig. 8). We investigate the relative performance of *JointNIDS* by the FC model according to complexity by varying the number of hidden layers from 0 to 2. In the case of non-hidden layers, the operation ratio between *2s0h* and *1s0h* is computed, and for one and two hidden layers, the corresponding ratios are similarly computed. When the percentage of major attack classes increases from 0.1 to 0.95, *JointNIDS* becomes more beneficial in terms of model complexity. For the considered dataset, the majority of attacks contains 97% of samples. If one hidden layer is used, the complexity ratio between the *JointNIDS* and FC models is 0.663, meaning that *JointNIDS* reduces the operations by 33.7%, compared to the FC model of *1s1h*.

TABLE V
EFFECTS OF OVERLAPPING RATIO ON THE PERFORMANCE OF THE JOINT CLASSIFICATION MODEL

Ratio r_o	Parameters	Classification Accuracy (%)	E2E Delay (ms)
0	100	93.22	0.96
0.2	117	93.29	1.03
0.4	134	93.41	1.04
0.6	151	93.64	1.083
0.8	168	94.38	1.141
1.0 (*1s1h*)	185	94.17	1.23

We now investigate the effects of the overlapping ratio on the number of network parameters, classification accuracy, and E2E delay of the classification model (Table V). Model *2s1h* is used as the default architecture of the joint classification model. As the overlapping ratio changes from 0 to 0.8, the classification architecture becomes more complex with a higher number of parameters, thus resulting in gradually higher accuracy. When the overlapping ratio is 0.8, the classification accuracy almost reaches the upper bound performance, which is 94.54% in the case of *1h2h*. If the overlapping ratio is set to 1, it is equivalent to the fully-connected model *1s1h*. In terms of E2E delay, due to the increasing trend in model complexity, we have higher latency for a packet to be classified on the way from the sender to a receiver host. In Table V, we observe a balance between classification accuracy and E2E delay with an overlapping ratio of 0.6, as being used as the default ratio in the model.

We also examine the effect of the number of hidden layers in *JointNIDS* in Fig. 9. As the number of hidden layers varies from 0 to 4 with the two switch scenario, the accuracy increases, while its resulting computational complexity linearly increases. We find a good trade-off point where the number of hidden layers is 1, showing a stable accuracy performance with a relatively decent computational overhead.

We have conducted experiments to verify the impact of λ value on the classification accuracy of the proposed model and the percentage of use of switch 1. The results are shown in Fig. 10, in which λ varies from 0.01 to 0.99. As λ increases, the classification accuracy of the decomposed model can be enhanced, because the outputs of both switches are used more frequently. The model complexity can be reduced by using only switch 1. As in Fig. 10, we have observed that the classification accuracy becomes stable, while the amount of data traffic classified by using only switch 1 decreases greatly when λ exceeds 0.7. For other experiments, $\lambda = 0.5$ is selected as a good trade-off point to guarantee the classification accuracy while maintaining low model complexity.

We measure the average delay at each switch and also the E2E delay from the sender to the receiver for various classification models as in Table VI. The delay includes the time for parsing, match-action, deparsing, and classification execution. Since SVM, DT, *1s0h*, and *1s1h* do not require the secondary switch for classification, only the generic parsing, match-action, and deparsing time is measured. The classification-related execution time is highlighted in bold text. For the same number of hidden layers, the joint classification model produces a lower E2E delay than its fully-connected complete model executed at a single switch. For example, model *2s1h* reduces the E2E delay by 0.185 ms, compared to the model *1s1h*. Other machine learning approaches, such as SVM and DT [22], have a slightly lower E2E delay, but with a relatively lower classification result, as shown in Fig. 7.

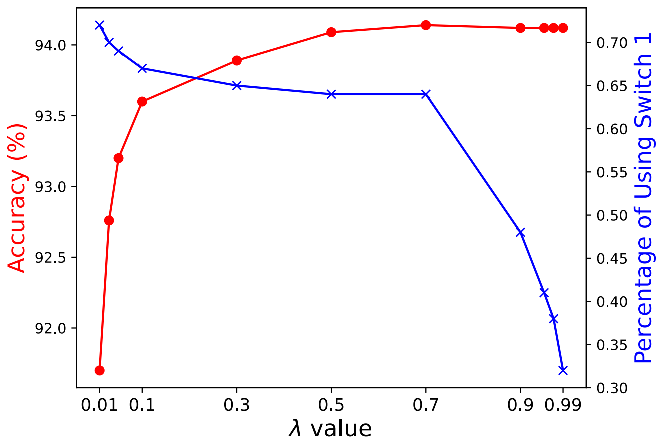

Fig. 10. Impact of threshold value λ on the performance of the proposed model. There is a trade-off between accuracy and computational complexity for predicting attack traffic classes.

TABLE VI
COMPARISON OF AVERAGE DELAY AMONG CLASSIFICATION MODELS

Classification Model	Average Delay for Single Packet (ms)		
	Primary Switch	Secondary Switch	E2E Delay
SVM	**0.62**	0.402	1.022
DT	**0.476**	0.437	0.913
1s0h	**0.556**	0.356	0.912
1s1h	**0.849**	0.419	1.268
2s0h	**0.450**	**0.334**	0.784
2s1h	**0.702**	**0.381**	1.083

D. Optimization of Traffic Management

To evaluate the effectiveness of our optimization scheme for maximizing traffic management, we use the empirical performance statistics in classification accuracy and delay, as reported in Section V-C. In an IoT network of five P4-supported switches with the traffic classification function, the number of packets originally sent to each switch is randomly selected in the range [100, 2,500], with one packet of 1,000 bytes. To evaluate the performance, the number of classified packets and the completion ratio of the total number of the classified packets out of the total number of the generated packets are quantified. Since the raw dataset is highly unbalanced, we have chosen a subset of the dataset, so that each attack class is distributed more evenly. Four out of 42 trace files are selected with 25,747, 4,980, 1,630, 5,000, and 1,244 samples in the normal, DoS, MitM, botnet, and reconnaissance classes, respectively. Then, the generated packets are classified by one out of six models: SVM, DT, *1s0h*, *1s1h*, *2s0h*, or *2s1h* on the selected subset, leading to the

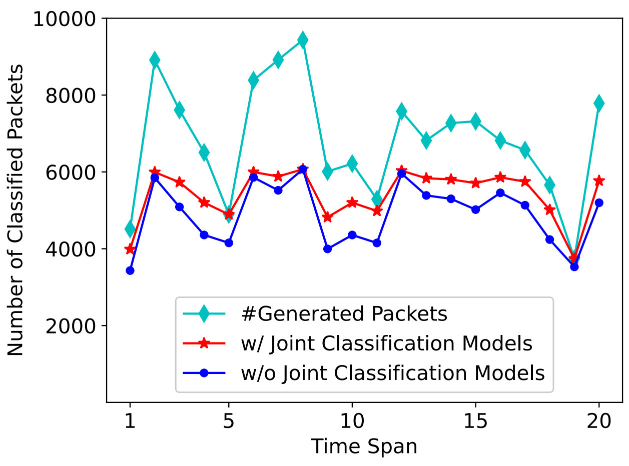

Fig. 11. Effects of joint classification models with different numbers of generated packets. Embedding our joint classification approach in NIDS improves the inspection throughput.

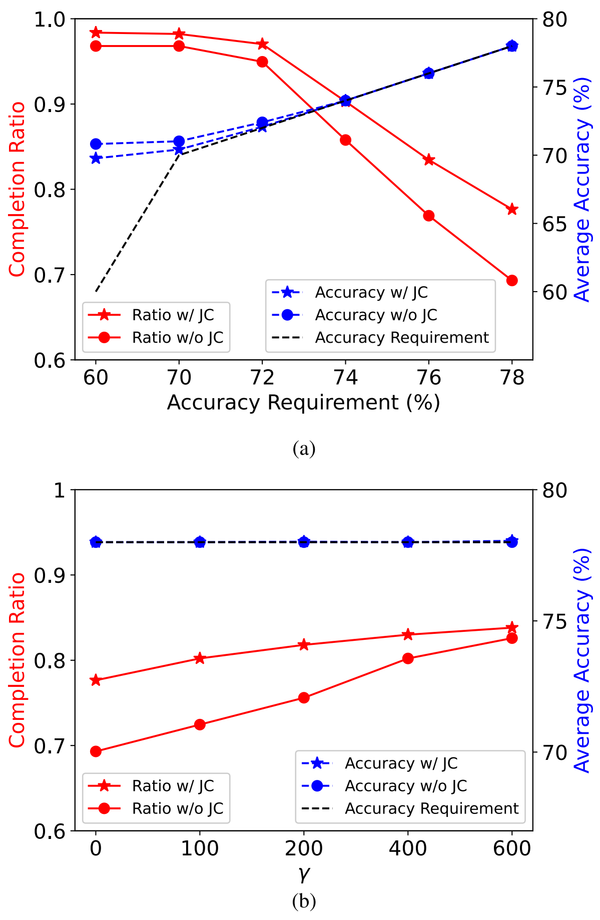

Fig. 12. Effect of different requirement constraints with and without joint classification. More traffic packets have been inspected with the help of *JointNIDS*.

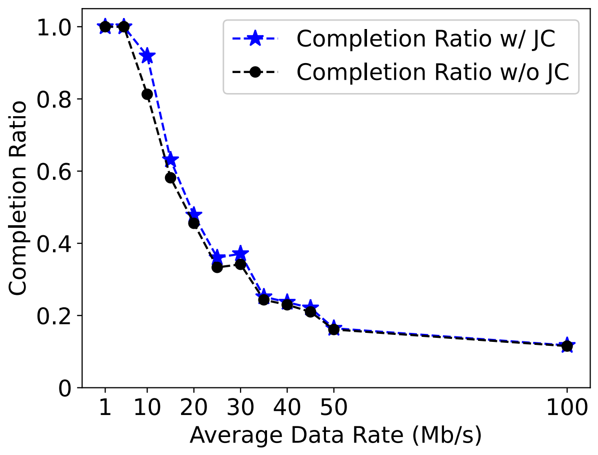

Fig. 13. Effect of joint classification models with different average data rates. Given a constrained network bandwidth, joint classification effectively inspects more traffic.

classification accuracy of 58.67%, 69.45%, 68.04%, 78.16%, 68.04%, and 80.14%, respectively.

Given the empirical performance metrics of accuracy and E2E delay, the optimal solution to the ILP problem is obtained by using a Python-based library called `pywraplp` with the SCIP (Solving Constraint Integer Programs) solver [1]. SCIP with a branch-and-bound approach divides the problem into subproblems, usually splitting the variable space into two disjoint parts. The subproblem splitting ends when the subproblem is infeasible, or when an optimal solution or no better solution for the subproblem is found.

We run the optimization of traffic management 20 times, generating random packets at each switch for the following comparison: without versus with our joint classification scheme as in Fig. 11. The joint classification-free models are listed as SVM, DT, *1s0h*, and *1s1h*. As the number of generated packets varies from 3,741 to 9,435, the number of classified packets with joint classification is higher than that of the one without joint classification. This means that the joint classification approach enables the general classification models to accept more incoming traffic, up to 21%, by effectively distributing traffic processing demands to another switch. Due to the limited computational resources at the switches, models with and without joint classification do not fully process the given incoming packets. The number of classified packets can not exceed 6,100 packets. Although there should be a limit in processing the simultaneously incoming packets at the edge, the joint classification-driven models improve the processing capability almost up to the performance limit.

We further investigate the traffic processing capability with various accuracy requirements by quantifying the completion ratio, depending on whether the joint classification is included or not, as shown in Fig. 12. As the first requirement, the requirement for a system to achieve a certain level of detection accuracy varies, for example, from 60% to 78%. The maximum number of packets offloaded to a different switch is denoted as γ and used as the second requirement.

We quantify the completion ratio and the end accuracy with respect to the accuracy requirement, while the maximum number of offloading packets γ is set to 0, as in Fig. 12(a). Packets randomly generated 10 times are injected at each switch, and the average performance is measured. As the system enforces the accuracy requirement from 60% to 78%, the completion ratio with joint classification outperforms the ratio without it, while both schemes satisfy with the accuracy requirement. For a given higher accuracy requirement, the joint classification-based optimization strategy has successfully processed even more traffic, with a performance gap of up to 8.3%.

This result is due to the fact that when the accuracy requirement is relatively low, both strategies use models with low classification performance to maximize the amount of data that can be managed. Thus, the difference in completion ratios between the two strategies is not significant. However, when the accuracy requirement is high, the models with high performance such as *1s1h*, and *2s1h* must be incorporated. Since the joint classification-based models require a lower delay for traffic classification – more packets can be processed for a given time – the use of joint classification models is more beneficial. For example, if the accuracy requirement is set to 78%, the completion ratios with and without joint classification models are 0.776 and 0.693, respectively. Fig. 12(b) shows the performance of both strategies with the different numbers of packets offloaded to switch i, given the accuracy requirement is set to 78. It is assumed that $\forall i \quad \gamma_i = \gamma$, where γ varies in the range [0, 600]. Intuitively, when γ becomes larger, switches can distribute the classification demand and exchange more packets. This approach makes more efficient use of computing resources in the whole network, and thus, the completion ratio in both cases increases. The performance gap in both strategies is reduced if γ increases, because more packets can be classified, even by joint-less classification models by the neighboring switches. For example, the gap between the two strategies is 0.075 and 0.01 when γ changes from 0 to 600.

Finally, we investigate the way in which the data rate affects the completion ratio in both traffic management strategies with respect to the average data rate at each switch, from 1 to 100 Mb/s. The accuracy requirement and γ are set to 78% and 0, respectively. If the average traffic rate at each switch is less than 5 Mb/s, both strategies are able to classify almost every generated packet. When there are more incoming data from each switch, the completion ratio in both cases gradually drops. Because there is insufficient computing resource to classify all of the incoming packets. Thanks to our joint classification approach, a strategy has helped to handle more traffic than joint inspection-free learning in a given same IoT network. This means that the approach contributes to ensuring more security and reliability for the network.

VI. Discussion

A. Effect of Environmental Constraints

The joint classification approach operates more effectively in a relatively heavy traffic environment. Heavy traffic conditions often arise from intensive packet injection into several switches, or network bandwidth shortage. A sequential packet inspection architecture operates collaboratively along a series of sequential switches. It allows selective turning of each switch to be active or inactive, for efficient usage of numerous IoT switches. By defining a unique inspection role for different attack types for each dedicated switch, the entire network finds a way to improve packet inspection throughput under the heavy traffic conditions. As the network traffic dynamics change, the adaptive adoption of joint classification and traffic management optimization in the data plane can contribute to defense against potential threats from arbitrary hacking attempts.

B. Extension to Joint Switch Group

A distributed packet inspection architecture with sequentially connected network switches offers performance improvement. To decide upon the effective number of joint switches, we may need to consider the following aspects of transmission delay between switches and the computation delay at each switch. Since transmission delay is not negligible in a complicated network topology with a relatively large number of network hops, the benefit of selective computation with collaboration could be compromised with the network overhead. In an environment with a specific application type, network bandwidth, and computing resource specification, there exists an effective operating point, suggesting the effective number of sequential switches for the purpose.

VII. Conclusion

With the aim of developing an on-device network intrusion detection system with low complexity for programmable edge devices, we propose a joint classification architecture *JointNIDS*. This architecture consists of a series of partially overlapping sub-models that are distributed sequentially over a primary switch and multiple secondary switches. The primary switch mainly detects major traffic classes, while the secondary switch takes over if necessary. *JointNIDS* architecture uses some shared overlapping sub-neural networks, to handle the remaining minor traffic classes. *JointNIDS* eventually reduces NIDS model complexity as well as detection time.

Our experimental results on the data plane in the P4 framework illustrate the effectiveness of *JointNIDS*, which produced lower detection delay without much sacrifice of classification accuracy, compared to counterpart algorithms. To demonstrate another benefit of *JointNIDS*, we propose an optimization algorithm for efficient traffic management. It maximizes the traffic management by all of the available switches, for given performance constraints of resource and communication overhead. The evaluation of the performance of various scenarios demonstrates that a collaborative architecture based on joint classification enables us to produce a better traffic management strategy, with the low transmission cost and high accuracy requirement.

JointNIDS allows the customization of the inspection of target attack types in NIDS. If a network operator wants to focus on detecting some specific network attacks with a given delay constraint, the primary switch can perform the meticulous inspection of these attack types. In future work, it would be interesting to design a heuristic packet assignment algorithm that can quickly find an optimal assignment for the system, particularly in a larger network. More extensive real-world evaluation of *JointNIDS* could also be conducted. Also, *JointNIDS* can be combined with model compression techniques such as neuron pruning and distillation, to construct an even more lightweight yet efficient detection system.

REFERENCES

[1] T. Achterberg, "SCIP: Solving constraint integer programs," *Math. Program. Computation*, vol. 1, no. 1, pp. 1–41, 2009.

[2] Z. Ahmad, A. S. Khan, C. W. Shiang, J. Abdullah, and F. Ahmad, "Network intrusion detection system: A systematic study of machine learning and deep learning approaches," *Trans. Emerg. Telecommun. Technol.*, vol. 32, no. 1, 2021, Art. no. e4150.

[3] I. Basicevic, N. Blazic, and S. Ocovaj, "On the use of principal component analysis in the entropy based detection of denial-of-service attacks," *Secur. Privacy*, vol. 5, no. 2, 2022, Art. no. e193.

[4] P. Bosshart et al., "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, 2014.

[5] T.-N. Dao and H. Lee, "Stacked autoencoder-based probabilistic feature extraction for on-device network intrusion detection," *IEEE Internet Things J.*, vol. 9, no. 16, pp. 14438–14451, Aug. 2021.

[6] D. Ding, M. Savi, G. Antichi, and D. Siracusa, "An incrementally-deployable P4-enabled architecture for network-wide heavy-hitter detection," *IEEE Trans. Netw. Service Manag.*, vol. 17, no. 1, pp. 75–88, Mar. 2020.

[7] D. Ding, M. Savi, and D. Siracusa, "Tracking normalized network traffic entropy to detect DDoS attacks in P4," *IEEE Trans. Dependable Secure Comput.*, 2021.

[8] F. Erlacher and F. Dressler, "FIXIDS: A high-speed signature-based flow intrusion detection system," in *Proc. IEEE/IFIP Netw. Operations Manage. Symp.*, 2018, pp. 1–8.

[9] F. Erlacher and F. Dressler, "On high-speed flow-based intrusion detection using snort-compatible signatures," *IEEE Trans. Dependable Secure Comput.*, vol. 19, no. 1, pp. 495–506, Jan./Feb. 2022.

[10] D. Narayanan et al., "PipeDream: Fast and efficient pipeline parallel DNN training," in *Proc. 27th ACM Symp. Operating Syst. Princ.*, 2019, pp. 1–15.

[11] R. Harrison, Q. Cai, A. Gupta, and J. Rexford, "Network-wide heavy hitter detection with commodity switches," in *Proc. Symp. SDN Res.*, New York, NY, USA, 2018, Art. no. 8.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2015, pp. 1026–1034.

[13] R. Hofstede, V. Bartoš, A. Sperotto, and A. Pras, "Towards real-time intrusion detection for NetFlow and IPFIX," in *Proc. IEEE 9th Int. Conf. Netw. Service Manage.*, 2013, pp. 227–234.

[14] Y. Huang et al., "GPipe: Efficient training of giant neural networks using pipeline parallelism," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, Art. no. 10.

[15] H. Kang, D. H. Ahn, G. M. Lee, J. D. Yoo, K. H. Park, and H. K. Kim, "IoT network intrusion dataset," IEEE Dataport, 2019, doi: 10.21227/q70p-q449.

[16] Y. Kang et al., "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *SIGARCH Comput. Archit. News*, vol. 45, no. 1, pp. 615–629, Apr. 2017.

[17] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proc. 9th ACM SIGCOMM Workshop Hot Topics Netw.*, New York, NY, USA, 2010, Art. no. 19.

[18] E. L. Lawler, "Fast approximation algorithms for knapsack problems," *Math. Operations Res.*, vol. 4, no. 4, pp. 339–356, 1979.

[19] B. Lewis, M. Broadbent, and N. Race, "P4ID: P4 enhanced intrusion detection," in *Proc. IEEE Conf. Netw. Function Virtualization Softw. Defined Netw.*, 2019, pp. 1–4.

[20] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: An ensemble of autoencoders for online network intrusion detection," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2018, pp. 1–15.

[21] N. Moustafa and J. Slay, "UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," in *Proc. IEEE Mil. Commun. Inf. Syst. Conf.*, 2015, pp. 1–6.

[22] F. Musumeci, V. Ionata, F. Paolucci, F. Cugini, and M. Tornatore, "Machine-learning-assisted DDoS attack detection with P4 language," in *Proc. IEEE Int. Conf. Commun.*, 2020, pp. 1–6.

[23] G. K. Ndonda and R. Sadre, "A two-level intrusion detection system for industrial control system networks using P4," in *Proc. 5th Int. Symp. ICS SCADA Cyber Secur. Res.*, 2018, pp. 31–40.

[24] R. N. Carvalho, J. L. Bordim, and E. A. P. Alchieri, "Entropy-based dos attack identification in SDN," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops*, 2019, pp. 627–634.

[25] D. Pisinger, "A minimal algorithm for the bounded knapsack problem," in *Proc. Int. Conf. Integer Program. Combinatorial Optim.*, Berlin, Heidelberg, 1995, pp. 95–109.

[26] Y. Qian, W. You, and K. Qian, "Openflow flow table overflow attacks and countermeasures," in *Proc. IEEE Eur. Conf. Netw. Commun.*, 2016, pp. 205–209.

[27] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *Proc. Int. Conf. Inf. Syst. Secur. Privacy*, 2018, pp. 108–116.

[28] C. Sinclair, L. Pierce, and S. Matzner, "An application of machine learning to network intrusion detection," in *Proc. IEEE 15th Annu. Comput. Secur. Appl. Conf.*, 1999, pp. 371–377.

[29] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, and J. Rexford, "Heavy-hitter detection entirely in the data plane," in *Proc. Symp. SDN Res.*, New York, NY, USA, 2017, pp. 164–176.

[30] L. Tang, Q. Huang, and P. P. C. Lee, "A fast and compact invertible sketch for network-wide heavy flow detection," *IEEE/ACM Trans. Netw.*, vol. 28, no. 5, pp. 2350–2363, Oct. 2020.

[31] P. R. Torres, A. García-Martínez, M. Bagnulo, and E. P. Ribeiro, "An elephant in the room: Using sampling for detecting heavy-hitters in programmable switches," *IEEE Access*, vol. 9, pp. 94122–94131, 2021.

[32] L. D. Tsobdjou, S. Pierre, and A. Quintero, "An online entropy-based DDoS flooding attack detection system with dynamic threshold," *IEEE Trans. Netw. Service Manag.*, vol. 19, no. 2, pp. 1679–1689, Jun. 2022.

[33] M. F. Umer, M. Sher, and Y. Bi, "Flow-based intrusion detection: Techniques and challenges," *Comput. And Secur.*, vol. 70, pp. 238–254, 2017.

[34] M. Wang, K. Zheng, Y. Yang, and X. Wang, "An explainable machine learning framework for intrusion detection systems," *IEEE Access*, vol. 8, pp. 73127–73141, 2020.

[35] J. Yoon, Y. Byeon, J. Kim, and H. Lee, "EdgePipe: Tailoring pipeline parallelism with deep neural networks for volatile wireless edge devices," *IEEE Internet Things J.*, vol. 9, no. 14, pp. 11633–11647, Jul. 2022.

[36] X. Zhang, L. Cui, F. P. Tso, and W. Jia, "pHeavy: Predicting heavy flows in the programmable data plane," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 4, pp. 4353–4364, Dec. 2021.

Thi-Nga Dao received the B.S. degree in electrical and communication engineering from Le Quy Don Technical University, Hanoi, Vietnam, in 2013, the M.S. degree in computer engineering from the University of Ulsan, Ulsan, South Korea, in 2016, and the Ph.D. degree in computer engineering from the University of Ulsan, in 2019. Since July 2019, she has been a Lecturer with the Faculty of Radio-Electronic Engineering, Le Quy Don Technical University. She was a Postdoctoral Fellow with the Computer Science and Engineering Department, Ewha Womans University, Seoul, South Korea, in 2021. Her research interests include machine learning-based applications in network security, network intrusion detection and prevention systems, human mobility prediction, and mobile crowdsensing.

HyungJune Lee received the B.S. degree in electrical engineering from Seoul National University, Seoul, South Korea, in 2001, and the M.S. and Ph.D. degrees in electrical engineering from Stanford University, Stanford, CA, USA, in 2006 and 2010, respectively. He joined Broadcom as a Senior Staff Scientist for working on research and development of 60 GHz 802.11ad SoC MAC. Also, he worked for AT&T Labs as a Principal Member of Technical Staff with the involvement of LTE overload estimation, LTE-WiFi interworking, and heterogeneous networks. He is currently an Associate Professor with the Computer Science and Engineering Department, Ewha Womans University, Seoul, South Korea. His research interests include distributed edge learning and future wireless networks based on machine learning-driven network system design.