2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS) | 978-1-7281-7002-2/20/$31.00 ©2020 IEEE | DOI: 10.1109/ICDCS47774.2020.00064

A Learning Approach with Programmable Data Plane towards IoT Security

Qiaofeng Qin, Konstantinos Poularakis, and Leandros Tassiulas
Department of Electrical Engineering and Institute for Network Science, Yale University, USA

***Abstract*—Security threats arising in massively connected Internet of Things (IoT) devices have attracted wide attention. It is necessary to equip IoT gateways with firewalls to prevent hacked devices from infecting a larger amount of network nodes. The match-and-action mechanism of Software Defined Networking (SDN) provides the means to differentiate malicious traffic flows from normal ones, which mirrors the past firewall mechanisms but with a new flexible and dynamically reconfigurable twist. However, vulnerabilities of IoT devices and heterogeneous protocols coexisting in the same network challenge the extension of SDN into the IoT domain. To overcome these challenges, we leverage the high level of data plane programmability brought by the P4 language and design a novel two-stage deep learning method for attack detection tailored to that particular language. Our method is able to generate flow rules that match a small number of header fields from arbitrary protocols while maintaining high performance of attack detection. Evaluations using network traces of different IoT protocols show significant benefits in accuracy, efficiency and universality over state-of-the-art methods.**

I. Introduction

Internet of Things (IoT) interconnects a multitude of devices interfacing with the physical world as sensors and actuators, facilitating their communication towards accomplishing assigned tasks. In such networks with massively interconnected devices, security is a major concern. A large amount of insecure IoT devices have become targets of botnet attacks [1], leading to some of the most potent DDoS attacks in history. IoT devices are vulnerable to more types of attacks compared with other devices [2], such as network attacks in different protocols (e.g., RFID, Zigbee, 6LoWPAN) and even physical attacks. Therefore, it has been a big challenge to guarantee the security of an IoT network.

Traditional methods to secure an IoT device require the deployment of *physical* and *application layer* protection in it, e.g., by strengthening the authentication and encryption during data transmission. However, such approaches usually involve firmware and even hardware modifications, taking a relatively long time period. Devices in which security policies are not updated in time will increase the risk of being hacked and becoming sources of infection to other devices. To prevent malware from spreading, *network layer* security approaches are also necessary. For example, firewalls can be deployed at IoT gateways, monitoring and separating malicious from normal traffic, as depicted in Figure 1.

This publication was supported partly by the National Science Foundation under Grant CNS 1815676, the Army Research Office under Agreement Number W911NF-18-10-378, and the Office of Naval Research.

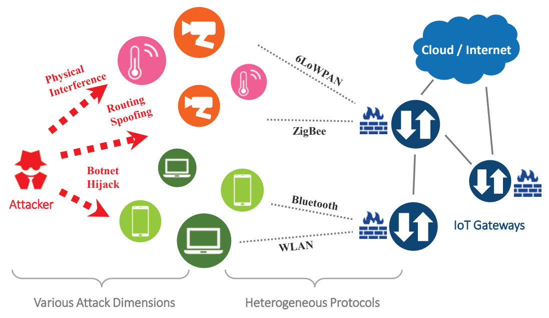


Figure 1. Firewalls deployed at IoT gateways targeting various types of attacks in heterogeneous protocols.

Software Defined Networking (SDN) provides a flexible framework for network management and is widely adopted in IoT networks. This flexibility can be exploited for the development and dynamic reconfiguration of network layer security mechanisms. By separating control and data planes, SDN protocols such as OpenFlow [3] make it possible to develop such mechanisms in a logically centralized and programmable manner. OpenFlow-enabled switches process incoming packets through match-and-action flow rules received from the controller checking specific header fields (e.g., MAC and IP addresses, TCP port, etc.) and performing actions such as forwarding or dropping accordingly.

A firewall can be developed by generating flow rules through *machine learning* algorithms, which have been demonstrated as a promising method for identifying attacks from even unknown or encrypted traffic flows [4]. However, this method presents several limitations. Specifically:

1) *Limitations in Learning Models*. The training features used by the machine learning algorithm are often the specific header fields of the packet. However, heterogeneous IoT protocols may have distinct packet header structures, leading to a problem that the feature extraction process and even the whole learning algorithm should be specifically *redesigned* for every different protocol. Besides, the *manual* feature extraction adds difficulty to achieve optimal performance.
2) *Limitations in OpenFlow*. The match fields of OpenFlow are *predefined* and *fixed*. Many IoT headers cannot be parsed by it, e.g., compressed IPv6 headers in 6LoWPAN packets, or application layer protocols such as MQTT and RESTful API. As a result, no proper flow

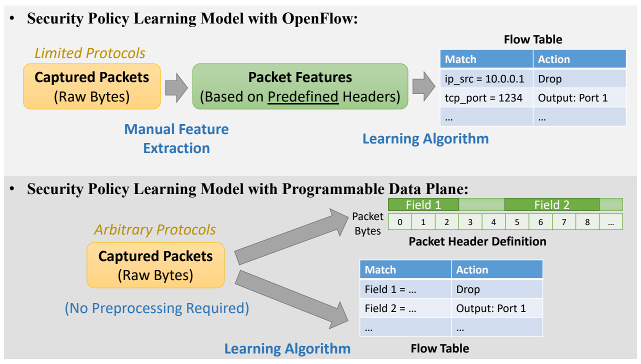

• Security Policy Learning Model with OpenFlow:
Limited Protocols
Captured Packets (Raw Bytes)
Packet Features (Based on Predefined Headers)
Flow Table
Match | Action
ip_src = 10.0.0.1 | Drop
tcp_port = 1234 | Output: Port 1
Manual Feature Extraction
Learning Algorithm
• Security Policy Learning Model with Programmable Data Plane:
Field 1
Field 2
Packet Bytes
0 1 2 3 4 5 6 7 8 …
Packet Header Definition
Arbitrary Protocols
Captured Packets (Raw Bytes)
Match | Action
Field 1 = … | Drop
Field 2 = … | Output: Port 1
(No Preprocessing Required)
Learning Algorithm
Flow Table

Figure 2. The learning process based on OpenFlow method and P4 language.

rules can be created in these cases. Although OpenFlow can be extended with user-defined headers by OpenFlow Extensible Match (OXM), it has limited functionality and hardware support in the above scenarios.

P4 language [5] provides possible solutions to the above challenges. Unlike OpenFlow which focuses on the control plane (i.e., the controller), P4 makes the data plane (i.e., the switches) programmable as well. Specifically, the packet headers are customizable by operators with the position and width provided, and table lookup can be conducted on these newly defined headers by the switches. This feature is especially meaningful in IoT scenarios, where support of different IoT protocols can be added by defining their headers [6].

Motivated by the above, we propose a new framework for IoT security and a corresponding learning algorithm which take advantage of the P4 language. Figure 2 illustrates its differences compared with the existing OpenFlow-based methods. The proposed method operates in two stages. In Stage 1, a learning algorithm trains *a dilated Convolutional Neural Network (Dilated CNN)* with raw packet bytes, skipping the step of manual feature extraction. In Stage 2, *a proper set of header field definitions* is inferred from the trained neural network, based on which flow rules for blocking traffic (dropping packets) are generated and installed in the IoT gateway (data plane switch). This method is applicable to heterogeneous IoT protocols. Besides, it is designed to take the constraints of switch memory cost and packet processing speed into consideration, realizing a trade-off between accuracy and efficiency.

The contributions of this work can be summarized as follows:

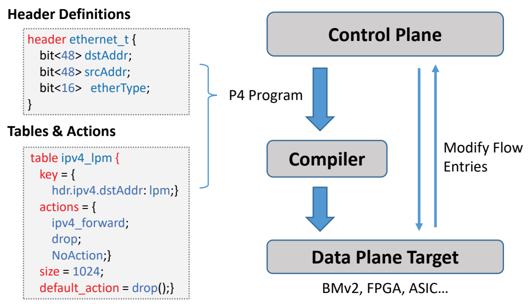
- **IoT Security Framework**. We propose a new framework for securing IoT networks and devices. Taking advantages of the programmable data plane of P4 language, we aim at developing a universal, highly accurate and efficient solution to identify malicious traffic flows of multiple IoT protocols.
- **Learning Algorithm (Stage 1)**. We propose a learning algorithm that trains a dilated Convolutional Neural Network (CNN) with raw packet bytes to set up a traffic classifier. This approach skips the step of manual feature extraction of OpenFlow based methods and thus requires minimum data preprocessing.
- **Header Field Definition (Stage 2)**. We develop a method for converting the abstract features learned in the trained CNN into a particular set of header fields, so that a proper set of flow rules can be installed at the IoT gateway. This way, the classification can be realized as a switch function at the IoT gateway for lower memory cost and faster processing speed.
- **Experimental Datasets**. We conduct experiments to create our own new datasets of IoT traffic and multiple types of attacks. With them as well as publicly available datasets, we evaluate the performance of the proposed framework and algorithm in all aspects. The results show that our method makes proper choices of header fields achieving a better attack (intrusion) detection accuracy level than state-of-the-art OpenFlow based methods (performance) while being also able to handle heterogeneous IoT protocols (universality). At the same time, the line speed of packet processing is maintained (efficiency).

The rest of the paper is organized as follows. Section II reviews our contribution compared to the related works while Section III presents our IoT security framework. Sections IV and V define and solve the header field definition problem based on the constructed CNN. The experimentation results are presented in Section VI, while we conclude our work in Section VII.

II. Related Work

Security problems of IoT devices have attracted wide attention. [2] and [7] provide comprehensive surveys of IoT attacks and classify them into various types. New types of attacks different from traditional networks threat IoT security, including a variety of attack methods in IoT protocols such as Zigbee and 6LoWPAN [8], [9], [10], as well as physical attacks targeting the sensors and actuators [11], [12]. These works suggest adding authentication mechanisms to the devices. However, a network-level security solution is also necessary for preventing malware from spreading among vulnerable IoT devices, such as botnets [13]. Our firewall implementation at the IoT gateway complements the device-level authentication for a more powerful security guarantee.

Network-level security approaches can be grouped into two categories. The first category applies machine learning methods on specific packet headers [14]. For example, [15] applies learning on 6LoWPAN headers. Kalis [16] provides knowledge-driven solution detecting IoT attacks, while DÏoT [17] and IoT Sentinel [18] identify the IoT device types by learning. Though these methods are effective, they usually require pre-knowledge from protocol definitions or device manufacturers. Due to the large diversity of IoT devices and protocols, we explore another direction leading to a more universal solution for heterogeneous IoT systems in case that such pre-knowledge is not available.



```
header ethernet_t {
   bit<48> dstAddr;
   bit<48> srcAddr;
   bit<16>  etherType;
}
```

```
table ipv4_lpm {
   key = {
      hdr.ipv4.dstAddr: lpm;}
   actions = {
      ipv4_forward;
      drop;
      NoAction;}
   size = 1024;
   default_action = drop();}
```

Figure 3. The protocol independence and reconfigurability of P4 language.

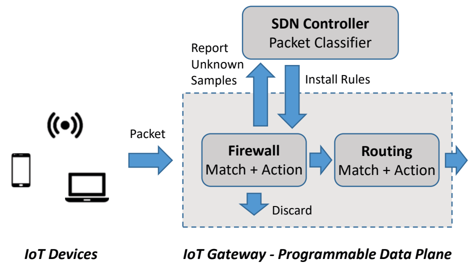


Figure 4. The control and data planes of the proposed framework, both programmable.

The second category classifies packets based on raw packet bytes rather than header fields. Machine learning methods, especially neural networks are also widely applied for it, such as [19], [4], [20]. These approaches have high accuracy and are not limited to specific protocol or device types. However, they can only be deployed in a remote server/host rather than a switch (IoT gateway). Therefore, packets cannot be processed at the line speed.

We aim at combining the merits of the two approaches above, developing intrusion detection as a switch function at the IoT gateway and at the same time not relying on assumptions of device and protocol types. Benefiting from their programmable, flexible and efficient packet processing capabilities, recent developments in SDN make the implementation of such switch function possible. For example, Sensor OpenFlow [21] and SDN-Wise [22] extend OpenFlow protocol in this direction. Besides, the programmable data plane brought by P4 [5] shows stronger capability in handling heterogeneous IoT protocols and helps researchers to explore further in this field. There is an increasing research interest in deploying and managing P4 switches. [23] aims at aggregating sensor data from multiple packets by P4 header operations. [6] achieves multi-protocol switching of IoT services by deploying P4-enabled switches. Our proposed IoT security framework and corresponding learning algorithm are also based on P4, which will be described in the next sections.

III. Security Framework

The proposed system has two components. The first part is the control plane, an SDN controller which is a software entity hosted in a node with sufficient computation capacity, e.g., a conventional cloud server or an edge cloud node. The second part is the data plane, which can be an IoT gateway. We consider the case that the IoT gateway is programmable by supporting the P4 language [5].

P4, or Programming Protocol-independent Packet Processors language is designed with reconfigurability and protocol independence. More specifically, the control plane (controller) is able to define how a data plane device (switch or gateway) parses a packet in a programmable and automated way (reconfigurability). First, one or more headers are defined as a list of fields given their positions and widths in bits. Then, a parser works as a state machine to extract headers, following a series of match+action tables, which is similar to OpenFlow, except that header fields are not predefined (protocol independence). The whole workflow is depicted in Figure 3.

As shown in [6], a P4-enabled gateway is capable of serving IoT devices of heterogeneous network protocols. Our aim is to use the IoT gateway to identify malicious incoming traffic flows (e.g., from a hijacked IoT device) before they are routed to other domains and devices. We program the IoT gateway to execute a firewall function before the routing function. The firewall keeps a match+action table recording the features of known packets, which are the values of certain packet header fields. These fields will be checked inside the incoming packets and marked as normal or malicious based on the flow rules installed in the table. Normal packets will be passed to the routing function without modifications. On the other hand, actions can be defined to handle the malicious packets, e.g., blocking them or forwarding them to a honeypot. The flow rules are generated by the SDN controller, where a classifier is deployed and responsible for judging whether a flow is malicious or not. The controller is able to convert classification results into header field definitions and flow rules to install them in the firewall at the IoT gateway either reactively or proactively. The whole architecture is depicted in Figure 4.

Two key problems are required to be solved in the proposed system. First, we need to find algorithms for classifying packets with high accuracy. Second, P4 match+action tables should be generated, making classification a data plane function which achieves line-speed packet processing. Besides, the solution we expect should be universal for heterogeneous IoT protocols, i.e., neither algorithm redesign nor protocol-dependent data preprocessing is required. In the next two sections, we will formally propose a formulation and a two-stage solution corresponding to the two key problems.

IV. Problem Modeling

Assumptions. To model the two problems, we consider a scenario of one IoT network domain equipped with one gateway along with its SDN controller. This scenario can be easily extended into a multi-domain or multi-gateway topology by deploying the same solution in each domain. The gateway is responsible for identifying attacks among all traffic flows going through it, so that it can block current and future packets of the attack flow to prevent it from spreading, e.g., a hijacked device outside the domain infecting devices inside the domain, and vice versa. We assume that the security of the gateway itself and its SDN controller is not compromised.

Packet Classification. The features that can be used for classifying network traffic can be divided into two types, the *packet-level features* (e.g., IP address, TCP port, payload length), and the *flow statistics* (e.g., packet count, duration). The programmable data plane of P4 brings opportunities for defining new packet-level features, not restricted to OpenFlow's pre-defined collection, which is particularly important for the IoT network where heterogeneous protocols coexist. Besides, previous studies like [19] claim several other merits of learning directly from packet bytes, including the higher accuracy and the ability to classify encrypted traffic. Therefore, our work is focused on the packet-level features type of classification.

We use the first N bytes of the packet as features for classification. The packet can be thus represented by a vector $\mathbf{x} = (x_1, x_2, ..., x_N)$ where each element $x_i \in [0, 1]$ $\forall i \leq N$ is a number converted from a byte. If the length of a packet is less than N, zero padding is applied. A classifier in the control plane should provide a function $F(\mathbf{x})$ judging the packet. We consider a binary output indicating whether the packet belongs to a normal traffic flow (i.e., $F(\mathbf{x}) = 0$) or a malicious one (i.e., $F(\mathbf{x}) = 1$). We can directly extend the method for multiple output values where the gateway takes different actions depending on the type of attack.

Header Fields Definition. While the control plane can check the bytes inside the packet one-by-one (and therefore compute the $F(\mathbf{x})$ value), such fine-grained classification may not be possible in the dataplane (IoT gateway) as this would require to install a huge number of flow rules for all possible combinations of the N bytes. This is not feasible since it would lead to unrealistic memory cost and latency of lookup and processing packets. Taking advantage of P4, any *substring* of packet bytes can be regarded as a *header field* by the gateway, based on which flow rules will be generated. Therefore, we can effectively limit the number and length of flow rules, as well as the associated packet processing latency, by carefully defining a small number of packet byte substrings as header fields at the gateway.

Formally, we define the *Header Fields Definition* $H = \{h_k, k = 1, 2, ..., K\}$ which is a set of K substrings of bytes. [24] investigates various P4-enabled devices to show that the number of header fields has an impact on the performance. Therefore, we require that $K \leq K_{max}$ where $K_{max} << N$ so that to ensure a maximum memory cost and packet processing latency requirement is met. Each element $h_k = (a_k, a_k + L_k)$ is a substring starting from the a_k-th byte of the packet and ending at the $(a_k + L_k - 1)$-th byte, with its length L_k. These substrings should not overlap with each other, i.e., $a_{k+1} \geq a_k + L_k$ for any k, to avoid wasting memory. Unlike the traditional definition of header fields, each of which contains a specific type of information (e.g., network address or port number), we do not restrict that every substring defined by our method corresponds to a clear entity. Instead, *we aim for an algorithm capable in learning the meaning and importance of different substrings*, so that it can minimize the requirement of data preprocessing and be applicable to heterogeneous IoT protocols.

Based on the Header Fields Definition H, the information actually extracted from a packet $\mathbf{x}$ is $\mathbf{x}^H = (x_{a_1}, ..., x_{a_1+L_1-1}, ..., x_{a_K}, ..., x_{a_K+L_K-1})$. Therefore, the packet classification executed at the gateway follows a different function from $F(\mathbf{x})$, which depends on the definition of header fields H. We denote this function by $F^H(\mathbf{x}^H)$. Our goal is to find proper H and $F^H(\mathbf{x}^H)$ functions which satisfy the constraints mentioned above and are able to predict the packet classification at a high accuracy.

V. Methodology

A. Overview

We solve the two problems specified in the previous section in *two stages* as depicted in Figure 5. In Stage 1, we build and train a *neural network (NN)* as the packet classifier. The training is based on raw packet bytes without considering the definition of header fields. This classifier will be deployed at the control plane. In Stage 2, we calculate *importance scores* for each possible substring of packet bytes using the information from the trained NN (Neuron Weights), and then select non-overlapping substrings with largest scores to be included in the header field definition, which will be installed at the gateway (data plane) along with a match+action flow table.

Initially, the NN is trained offline with captured network traces. The trained NN is then deployed at the controller as the packet classifier. For the data plane, both proactive and reactive operating modes are available according to different scenarios. In the first mode, the controller installs both header field definitions and corresponding flow rules from training data proactively at the gateway. The gateway can therefore process new incoming packets at the line speed without forwarding them to the controller. In the second mode, the controller can proactively install header field definitions only, and install flow rules in a reactive way by replying to the gateway's queries. This mode incurs less memory cost in the gateway but increases latency due to the controller-gateway communication each time when the gateway receives unknown packets.

After the initial offline training, with the gateway sampling new packets and sending them to the controller, the two-stage process can be repeated in an online manner optionally, as long as the labels of packets can be acquired by the controller as well. The controller can also dynamically update the header field definition by compiling a new P4 program. All these operations are supported by the P4 specification.

B. Stage 1: Neural Network Structure

We apply methods of supervised learning for the packet classification. In particular, trained with a labeled dataset (i.e.,

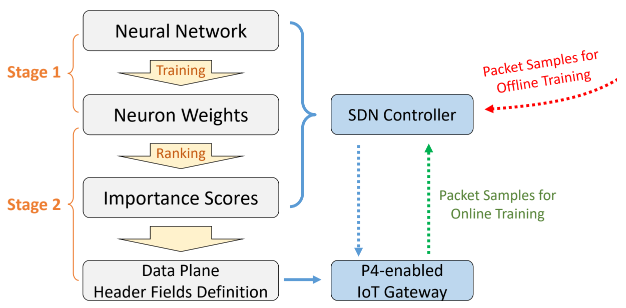


Figure 5. Illustration of the proposed two-stage learning approach. Packet classification is realized by the SDN control plane in Stage 1, followed by header field definition and implementation at the IoT gateway in Stage 2.

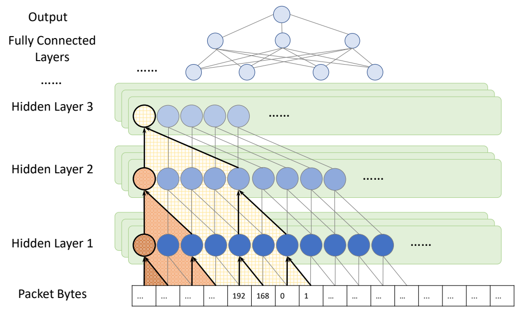


Figure 6. Structure of the dilated convolutional neural network (Dilated CNN) for packet classification.

large amount of packets marked as either malicious or normal), the classifier should be able to infer the expected output of a new input (the function $F(\mathbf{x})$). A Neural Network (NN) [25] is a computing system for supervised learning. It consists of several hidden layers and an output layer. Each layer is constructed by building blocks called neurons. For example, if we arrange the neurons of each layer in an array with index n (corresponding to the byte index of the packet), assign another index $i = 1, 2, ..., I_t$ for each layer t and take the packet byte vector $\boldsymbol{x}$ as the input, the output of a neuron in the first hidden layer is:

$$c^1_{ni} = f(\boldsymbol{w^{1;ni}} \cdot \boldsymbol{x} + \boldsymbol{b^{1;ni}}) \tag{1}$$

The output of each layer is the input of the next layer. For the neuron in the t-th hidden layer ($t > 1$), the output is:

$$c^t_{ni} = f(\boldsymbol{w^{t;ni}} \cdot \boldsymbol{c^{t-1}} + \boldsymbol{b^{t;ni}}) \tag{2}$$

where $\boldsymbol{w^{t;ni}}$ is a 2D vector of trainable weights, $\boldsymbol{b^{t;ni}}$ is a bias term, and f is a non-linear activation function.

Among various NN structures, we adopt 1D Dilated Convolutional Neural Network (Dilated CNN) [26] as depicted in Figure 6. In each hidden layer t, connections are local and dilated with step size 2^{t-1}. In other words, each neuron with index i only takes two rows of neurons with indices i and $i + 2^{t-1}$ in its last layer as the inputs. Neurons in the same layer share the same weight values. The output of the hidden layer neurons can be represented in the following way:

$$c^1_{ni} = f(w^1_\alpha \cdot x_n + w^1_\beta \cdot x_{n+1} + b^1) \tag{3}$$
$$c^t_{ni} = f(\boldsymbol{w^t_\alpha} \cdot \boldsymbol{c^{t-1}_n} + \boldsymbol{w^t_\beta} \cdot \boldsymbol{c^{t-1}_{n+2^{t-1}}} + \boldsymbol{b^t}), \forall t > 1 \tag{4}$$

where $\boldsymbol{w^t_\alpha}$ and $\boldsymbol{w^t_\beta}$ are two 1D vectors of trainable weights.

This structure brings two major benefits. First, for any hidden layer neuron c^t_{ni}, its inputs are limited in the range between packet bytes x_n and x_{n+2^t-1}, which means that we can establish a correspondence between a neuron c^t_{ni} and a substring $(n, n + 2^t)$ following the denotation in the last section. Second, the neuron receptive field is 2^t, increasing exponentially with the network depth. With T hidden layers, we can find neurons corresponding to any potential header field of length $2, 4, 8, ...,$ up to 2^T bytes. In other words, with a limited amount of layers, we are able to cover a wider range of packet substrings. This is beneficial in both representing the packet structure better and training the neural network more efficiently. After convolutional layers, we have fully-connected layers, the last of which has a single neuron taking the weighted sum of the last hidden layer outputs as the final result. This structure can be easily extended to multi-class classification, as long as we set up more neurons in the output layer.

C. Stage 2: Header Field Definition

In the next stage, we adopt a neural network pruning [27] technique to the trained network. Pruning compresses the neural network by reducing the number of neurons. With smaller memory and calculating costs, pruning facilitates the processing of NN in IoT scenarios [28], where the capacity of devices may be limited. However, besides this benefit, our main purpose is to deduct an optimal set of header field definition based on the results of pruning, therefore enabling the line-speed packet processing in a P4-enabled gateway.

Pruning leads to an *importance score* of each neuron. Neurons with higher importance scores play a more crucial role in the classification. According to [27], we apply the Inf-FS [29] algorithm to calculate the importance scores of neurons in the last hidden layer. Then, the importance scores are calculated for the remaining layers in a backpropagation manner.

Leveraging the one-to-one correspondence between neurons and header fields in the proposed CNN structure, we extend the notion of importance score from neurons to header fields. Unlike [27] that suggests to greedily select neurons with highest importance scores, our problem has additional constraints, e.g., that the header fields should not overlap with each other. Therefore, we propose a new problem formulation.

The input of the problem includes the importance scores of all neurons in each hidden layer t. We denote the importance score of neuron c^t_{ni} by s^t_{ni}. By summing these values, we denote the importance score of a potential header field $(n, n+2^t)$ by $S_n = \sum_i s^t_{ni}$. Then, we obtain the following optimization problem:

$$\max_{\boldsymbol{y}} \sum_{n=1}^{N^t} y_n * S_n \tag{5}$$

$$s.t. \sum_{n=1}^{N^t} y_n \leq K_{max} \tag{6}$$

$$y_n * y_{n+j} = 0, \quad \forall n < N^t, j < L \tag{7}$$

$$L = 2^t, \quad N^t = N - L + 1 \tag{8}$$

where $\boldsymbol{y} = (y_1, y_2, ..., y_{N^t})$ is the vector of variables to optimize, representing all possible substrings of length 2^t in the first N bytes of the packet. The binary element y_n indicates whether to select substring $(n, n + 2^t)$ in the header field definition ($y_n = 1$) or not ($y_n = 0$).

To solve this problem, we propose to use *Dynamic Programming* [30]. A *Bellman equation* can be easily defined based on two states; K as the amount of selected header fields and n_0 as the starting byte of the latest selected header field. We then have the following equations:

$$V(1, n_0) = S_{n_0}, \qquad \forall n_0 \leq N^t$$

$$V(K, n_0) = \max_{n+L \leq n_0} V(K-1, n) + S_{n_0}, \quad \forall n_0 \leq N^t, K > 1$$

Based on the above equations, any $V(K, n_0)$ value can be calculated by *recursion*. The maximum of our objective function is therefore $\max_{n_0 \leq N^t} V(K_{max}, n_0)$. As described in Algorithm 1, an optimal set of header fields H can be selected with reasonable $O(K_{max} * N)$ time complexity.

The parameters K_{max} (i.e., maximum number of header fields) and $L = 2^t$ (i.e., length of one header field) can be determined according to the capacity of different types of P4-enabled devices [24]. In general, a tradeoff between accuracy and cost can be achieved by adjusting these parameters. With fewer or shorter header fields, some different traffic flows may be regarded as the same one by the gateway, negatively affecting the classification accuracy. With more or longer header fields, however, it takes larger memory cost to store flow rules, and may slow down the packet processing in some implementations. In the next section, we will evaluate the exact impact of these parameters on different performance metrics.

Algorithm 1: Optimal Header Fields Selection

```
   Input: $S_1, S_2, ..., S_{N^t}, K_{max}, L$
1  for $n_0 \leq N^t$ do
2      $V(1, n_0) = S_{n_0}$;
3      $H(1, n_0) = \{(n_0, n_0 + L)\}$;
4  end
5  for $K = 2, 3, ..., K_{max}$ do
6      for $n_0 \leq N^t$ do
7          $n^* = \arg\max_{n+L \leq n_0} V(K-1, n)$;
8          $V(K, n_0) = V(K-1, n^*) + S_{n_0}$;
9          $H(K, n_0) = H(K-1, n^*) \cup \{(n_0, n_0 + L)\}$;
10     end
11 end
12 $n^* = \arg\max_{n \leq N^t} V(K_{max}, n)$;
   Output: $H = H(K_{max}, n^*)$
```

VI. Evaluation Results

To demonstrate the benefits of our P4-based IoT security approach, we perform evaluations using various real traffic datasets. We begin with presenting the datasets and algorithms that will be later used to generate the evaluation results.

A. Datasets & Algorithms

First, we use the following two publicly-available datasets of IoT network traffic:

- *ISCX Botnet 2014 Dataset [31].* This is a collection of botnet traffic traces from multiple well-known datasets. The types of traffic are mainly HTTP, P2P and IRC. This dataset is already divided into the training set and test set. The test set has more diversity than the training set, in order to evaluate the detection of unknown attacks. It is originally gathered for statistics-based classification and contains a huge amount of packets, therefore we sample 10% of the packets from each flow for packet-level training. We also randomly modify the IP fields because all malicious flows are remapped to fixed IP addresses in the original data.
- *CICAAGM Android Dataset [32].* This publicly available dataset captures the traffic of Android applications in real smartphones, including 250 adware, 150 malware and 1500 benign applications. Besides HTTP, there are also massive HTTPS traces, a large portion of which is SSL/TLS-encrypted. The raw packet bytes are available through PCAP files. We sample 1000 successive packets from each class of the trace for packet-level training and testing.

We also make our *own* efforts to create two *new* datasets using network simulators and real IoT devices we deploy, containing unique threats to IoT devices. These datasets contain protocols that OpenFlow cannot handle. On the contrary, we will demonstrate that P4 and our algorithm work well on them.

- *Cooja Network Simulator Dataset.* [33], [10] analyze different types of attacks in 6LoWPAN networks through the RPL routing protocol with the help of Contiki operating system and its Cooja simulator. Adopting similar methods, we run simulations of 10-node IoT networks with random topologies, and set up a malicious node conducting Version Number Attack and Increased Rank Attack. We collect packet bytes of both malicious and normal traffic flows to generate our dataset.
- *Waspmote IoT Sensor Dataset.* We also create a new dataset with measurements on real IoT devices (not simulator) we deploy. Specifically, we install temperature, humidity and luminosity sensors on a Waspmote [34] Smart Cities Pro sensor board. It periodically sends 802.15.4 frames to the gateway containing sensor data. If the electrical connection from a sensor to the board is impeded, the device will still send packets in the

same format but with the wrong values. This is indeed categorized as a physical attack on sensors rather than network attack. However, we will demonstrate that our method is also effective in detecting such unconventional attacks.

In each dataset except the first one, we randomly pick 80% of the samples for training, and the remaining 20% for testing. We implement several state-of-the-art algorithms and make comparisons with our method. In particular:

1) *Proposed P4-based Method.* In Stage 1, we build the deep neural network of the proposed structure with 4 convolutional layers each with 64 filters and the ReLU activation function [35], followed by two fully-connected layers with 100 and 50 neurons. At each hidden layer, a 0.05 dropout rate is set to avoid over-fitting. We keep the hyperameters unchanged when training with different datasets. In Stage 2, we produce the header field definition and install the corresponding flow rules to the IoT gateway.
2) *OpenFlow-based Methods.* As a comparison, we consider classification methods based on OpenFlow protocol, representing SDN without programmable data plane. We limit the features of classification within the predefined header fields of MAC, IP, TCP and UDP protocols according to the OpenFlow specification. As stated in [36], multiple machine learning techniques can be applied to these features, among which we choose two representative methods, Decision Tree (DT) and Support Vector Machine (SVM).
3) *1D Convolutional Neural Networks (1D-CNN).* We also consider other deep learning approaches for packet classification which (similar to our method) take packet bytes rather than some specific header fields as the input. We implement two 1D-CNN imitating the structures and hyperameters in [19], [4], denoted by CNN-1 and CNN-2. These CNNs provide the same type of output as our Stage 1 output. However, they are not capable in producing a header field definition as Stage 2 of our method does. In other words, the classification cannot be executed as a switch function for line-speed packet processing.

We implement DT and SVM models using scikit-learn [37] library, and implement NNs in TensorFlow [38]. To verify the header field definition calculated by our algorithm, we also conduct emulations with Mininet [39] and P4 behavioral model software switch (BMv2) [40]. The experiments are conducted on a desktop computer with Intel Core i7-7700 Processor, 16 GB RAM and GeForce GTX 1060 graphics card. We make our algorithm implementation and new datasets publicly available [41] for the benefit of research community.

B. Metrics

We evaluate the performance of the classification algorithms using as metric not only accuracy, but also precision and recall. We denote the number of correctly identified malicious packets by TP and incorrectly identified ones by FP. We denote the number of correctly identified normal packets by TN and incorrectly identified ones by FN. The metrics are calculated as follows:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{9}$$

$$precision = \frac{TP}{TP + FP}, \quad recall = \frac{TP}{TP + FN} \tag{10}$$

Considering that the datasets have uneven class distributions (where malicious samples account for around 30% in each dataset, except the Cooja dataset with around 10% malicious samples), we also calculate the F1 score defined as the harmonic mean of precision and recall:

$$F_1 = 2 * \frac{precision * recall}{precision + recall} \tag{11}$$

C. Classification (Stage 1) Performance

In this subsection, we evaluate Stage 1 of the proposed method. We compare the classification performance of the proposed dilated convolutional neural network with the two other CNN structures as well as with the DT and SVM OpenFlow-based methods.

Method	Accuracy	Precision	Recall	F_1
DT	0.790	0.694	0.659	0.676
SVM	0.773	0.706	0.544	0.615
CNN 1	0.907	0.897	0.816	0.854
CNN 2	0.909	0.903	0.816	0.857
Proposed	**0.911**	0.904	0.822	**0.861**

Table I
PERFORMANCE METRICS ON ISCX DATASET.

ISCX Botnet. We train and test all the algorithms on the ISCX dataset. Table I shows the accuracy of each algorithm. Compared with methods based on OpenFlow headers, the CNNs (including our approach) that take raw bytes as the input have significantly better performance. We also find that CNN-based methods outperform other algorithms in both precision and recall rates, leading to higher F1 score.

Method	Accuracy	Precision	Recall	F_1
DT	0.890	0.833	0.771	0.801
SVM	0.895	0.933	0.646	0.780
CNN 1	0.882	0.833	0.738	0.782
CNN 2	0.898	0.870	0.760	0.811
Proposed	**0.908**	0.927	0.736	**0.820**

Table II
PERFORMANCE METRICS ON CICAAFM DATASET.

CICAAGM dataset. We perform similar training and testing on the CICAAGM Android dataset, which contains a larger diversity of traffic flows including SSL/TLS encrypted ones. The results are depicted in Table II. Although the performance difference is not as large as in the ISCX dataset, our algorithm still achieves highest accuracy than the other algorithms. We note that while the SVM OpenFlow-based method reaches

Dataset	Cooja		Waspmote	
Method	Accuracy	F_1	Accuracy	F_1
CNN 1	0.998	0.991	0.995	0.993
CNN 2	0.994	0.971	0.998	0.996
Proposed	**0.995**	**0.973**	**1.00**	**1.00**

Table III
PERFORMANCE METRICS ON OTHER DATASETS.

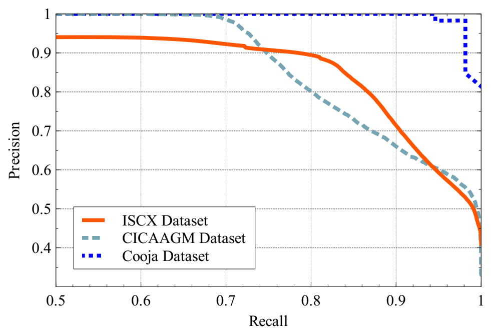


Figure 7. The precision-recall curve on different datasets.

higher precision, it severely degrades the recall value, leading to a lower F1 score.

Cooja dataset and Waspmote dataset. The Cooja and Waspmote datasets are relatively simple, each with smaller amount of samples and only two types of attacks. However, the former contains compressed 6LoWPAN headers, and the latter has abnormalities which can only be identified from the packet payload rather than the headers. Therefore, *the packets are not readable and can no longer be classified by the OpenFlow-based methods (i.e., DT and SVM).*

As shown in Table III, all three CNNs are capable of identifying the RPL routing attacks and sensor physical attacks with accuracy higher than 99%. The performance metrics of different methods are generally at the same level. Except being slightly worse than the CNN-1 in the Cooja dataset, our proposed network has superior performance in accuracy and F1 score. Especially, it achieves perfect predictions in the Waspmote dataset.

Performance Tradeoff. We are also interested in the trade-off between the different performance metrics. In some scenarios, the false alarms must be controlled, otherwise system failures can happen. To achieve this, we can apply a threshold to the CNN output. We depict the respective precision-recall curves for different thresholds in Figure 7 for all datasets except the Waspmote dataset where perfect predictions have been reached. We notice that in all datasets there is a space to increase the precision further at a cost of the recall.

Main Takeaways. (1) P4-based learning methods with packet bytes as the input can achieve better classification performance compared with OpenFlow-based learning methods that take as input predefined header fields. They can also handle heterogeneous protocols and application layer contents of packets, where OpenFlow-based methods are not applicable. (2) Our proposed Dilated CNN structure achieves similar or better performance than other state-of-the-art CNN approaches that take the same input (packet bytes).

D. Header Field Definition (Stage 2) Performance

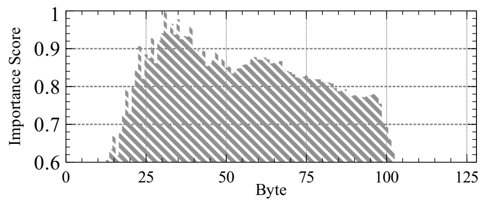
The classification performance benefits in the previous subsection are important but not surprising. It was expected that taking packet bytes rather than predefined headers as input to the learning algorithm achieves superior classification performance as the classifier design space is larger. Still, the above results quantified the exact performance improvement we can achieve and verified the suitability of our proposed Dilated CNN structure compared to other CNN structures.

The main contribution of our work, however, lies on the implementation of the intrusion detection function directly inside the data plane (P4-enabled IoT gateway). This is important because it enables line-speed packet processing that is not available in the other learning methods like CNN-1 and CNN-2. To achieve this, Stage 2 of our learning method uses the trained Dilated CNN to define a particular set of packet byte substrings as header fields that will be used by the gateway to install flow rules. Therefore, matched packets will be directly handled by the gateway without requiring to be forwarded to the SDN controller or another remote firewall function. In the sequel, we elaborate on the header field definition and corresponding classification performance achieved by Stage 2 of our algorithm.

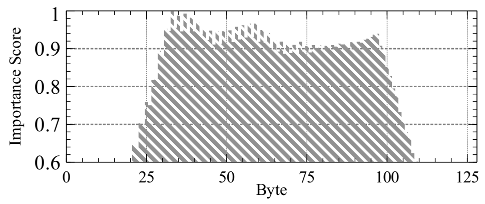
Profiles of Importance Scores. Following the procedure described in Section V-C, we calculate the importance scores for all substrings of length $1, 2, 4, 8$ and 16 in the first $N = 128$ byte positions. For example, Figure 8 depicts the results of importance scores (after normalization) for every single byte.

The profiles of the datasets show different and complicated tendencies. However, there are also some intuitive results:

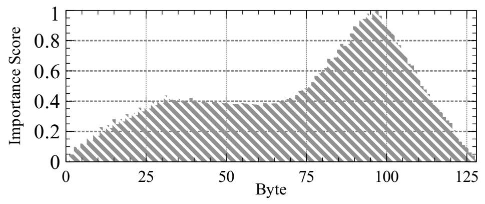
- *ISCX dataset* (with IP addresses masked). The algorithm highly scores both TCP/UDP fields and some positions in the application layer.
- *CICAAGM Android dataset*. The curve has three peaks in the IP address field, the TCP port field and application layer. This distribution implies that the classifier makes predictions based on information from headers of multiple network layers, which is an advantage of adopting SDN and P4. For example, in the case where the packet payload is SSL/TLS encrypted, even if the classifier is not able to parse application-layer information, it is able to make predictions based on TCP/IP headers with a high accuracy. On the other hand, the application-layer headers reveal much information in those packets without encryption.
- *Cooja dataset*. High importance score is given to 97-th byte. It is reasonable because all attacks occur through DODAG Information Object (DIO) messages [33] of 96 bytes. The algorithm takes the packet length into account when making classification.



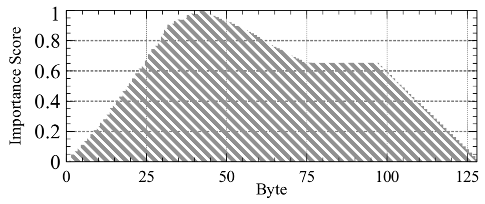
(a) ISCX Dataset.



(b) CICAAGM Dataset.

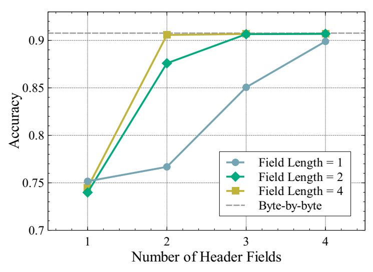


(c) Cooja Dataset.

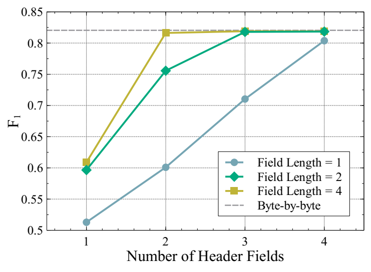


(d) Waspmote Dataset.

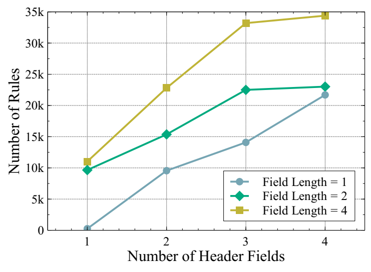
Figure 8. Distributions of single-byte importance scores in different datasets.



(a) Accuracy.



(b) F1 score.



(c) Memory cost.

Figure 9. Accuracy, precision and memory cost with different header fields selected in CICAAGM dataset.

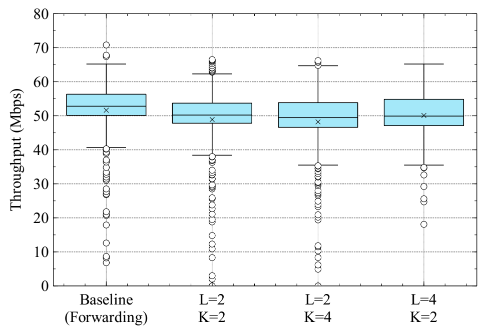


Figure 10. Throughputs with different header field definitions.

- *Waspmote dataset*. The algorithm successfully assigns highest importance scores to Byte 31, 32 and Byte 36, 37 in every 802.15.4 frame which store the the sensing data in question.

These distributions demonstrate that the importance scores calculated by our method successfully identify header fields that are crucial in classifying packets.

Impact of Header Fields on Accuracy. The proposed Dynamic Programming algorithm (Algorithm 1) will select as header fields the substrings of the packet bytes that have the highest importance scores. Taking the CICAAGM Android dataset as an example, Figure 9(a) and 9(b) show the accuracy and F1 score as we increase the number of header fields we match in the gateway node (K_{max} equal to 1, 2, 3 or 4) and for different header field lengths (L equal to 1, 2 or 4). The byte-to-byte approach corresponds to the packet classifier in Stage 1 of our method described in the previous subsection. Intuitively, the performance improves with the number of header fields. According to the results, it is not necessary to have a large number of header fields. *With three 2-byte-long fields or two 4-byte-long fields, the classification is almost as accurate as the byte-to-byte approach*. The difference is around 0.1% in accuracy values.

Impact of Header Fields on Costs. Next, we examine the costs associated with the header field definition, measured by the number of flow rules stored in the gateway node. Since more rules lead to a larger memory occupancy and more queries to the control plane, we need to keep their number as low as possible. Figure 9(c) shows that the number of rules required for classification increases with both the length and the number of header fields selected. Therefore, *a tradeoff exists between the accuracy and cost*. The balance point can be achieved by adjusting the values of K_{max} and L parameters in our algorithm. Notice that although the number of all possible values of a header field increases exponentially with its length, the growth is not drastic in practice. From the evaluation results, the tendency is closer to a linear growth.

We need to emphasize that the proposed intrusion detection mechanism does not incur much additional costs in other aspects such as the network latency and throughput, because it only adds an one-time table lookup in the packet processing procedure. To verify this intuition, we create a virtual network with one BMv2 switch and two hosts using the Mininet emulation platform. We implement several sets of header field definitions and flow tables similar to the results in Figure 9. We use this virtual network to measure the maximum throughput achieved by our mechanism for different K and L choices and compare it with the baseline L2 forwarding mechanism that does not perform any intrusion detection. The results are depicted in Figure 10. We notice that the maximum throughput is reduced by less than 10% compared with the baseline, i.e., the line speed of packet processing is maintained. In the same scenario, we have another approach that forces packets to go through an application-layer single-thread analyzer based on Scapy [42] before being forwarded, which represents the case if adopting solutions similar as CNN-1 and CNN-2 in the last subsection. In this case, no larger throughput than 1 Mbps is achieved. Therefore, it is extremely beneficial to implement the intrusion detection as a switch function inside the IoT gateway with the help of the programmable data plane feature.

Method	Optimal		Random	
# of Fields	Accuracy	F_1	Accuracy	F_1
1	0.740	0.597	0.689	0.325
2	0.876	0.757	0.765	0.490
3	0.907	0.818	0.775	0.557
4	0.907	0.818	0.802	0.639

Table IV
COMPARISONS BETWEEN THE PROPOSED ALGORITHM AND RANDOM SELECTED HEADER FIELDS. (THE LENGTH OF EACH FIELD IS 2 BYTE IN BOTH CASES.)

Optimal Selection of Header Fields. Last but not least, to demonstrate that the importance scores are proper metrics for the data plane definition, we compare the optimal selection of header fields in our algorithm with random selections. As shown in Table IV, with the same number of selected header fields, the performance of our algorithm is significantly better, with more than 10% accuracy and around 20% more F1 score than the random selection.

Main Takeaways. A similar level of packet classification accuracy as the byte-to-byte approach can be achieved by merely matching a small number (two or three) of header fields appropriately selected based on the importance scores in the associated neural network. When implemented as a P4 switch function at the IoT gateway, this approach requires low memory and latency cost and incurs small throughput loss for table lookup (less than 10%, i.e., line speed is maintained), while alternative application-layer intrusion detection mechanisms would cause a multi-fold throughput reduction to achieve the same level of functionality.

VII. CONCLUSION

In this paper, we studied new opportunities for enhancing security in the IoT network brought by the programmable data plane. Namely, we proposed a two-stage deep learning method based on P4 language that first trains a neural network as the packet classifier and in a later stage selects packet byte substrings as header fields and installs appropriate flow rules to realize intrusion detection functionality inside the IoT gateway. Evaluation results on publicly available and newly developed datasets of IoT scenarios demonstrated the performance benefits and universality of the proposed method compared with state-of-the-art OpenFlow-based methods. Importantly, the results verified that a more favorable tradeoff between detection accuracy, memory cost, latency and throughput can be achieved by the proposed method.

We believe that this paper opens exciting directions for future work. First, flow statistics can be taken into consideration along with the packet-level features to further enhance the accuracy and precision of the intrusion detection mechanism. Second, the classification decision in our model is based on exact matching of the packet headers. Flow table compression, that properly applies wildcard flow rules, constitutes another promising approach for improving further the efficiency and reducing the resource consumption costs. Last but not the least, we also plan to perform implementations and evaluations on different P4-enabled devices, such as hardware switches and SmartNICs, where higher performance of throughput and latency is expected.

REFERENCES

[1] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, "Ddos in the iot: Mirai and other botnets," *Computer*, vol. 50, no. 7, pp. 80–84, 2017.

[2] I. Andrea, C. Chrysostomou, and G. Hadjichristofi, "Internet of things: Security vulnerabilities and challenges," in *2015 IEEE Symposium on Computers and Communication (ISCC)*. IEEE, 2015, pp. 180–187.

[3] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008. [Online]. Available: http://doi.acm.org/10.1145/1355734.1355746

[4] M. Lotfollahi, M. J. Siavoshani, R. S. H. Zade, and M. Saberian, "Deep packet: A novel approach for encrypted traffic classification using deep learning," *Soft Computing*, pp. 1–14, 2017.

[5] 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.

[6] M. Uddin, S. Mukherjee, H. Chang, and T. Lakshman, "Sdn-based multiprotocol edge switching for iot service automation," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 12, pp. 2775–2786, 2018.

[7] F. A. Alaba, M. Othman, I. A. T. Hashem, and F. Alotaibi, "Internet of things security: A survey," *Journal of Network and Computer Applications*, vol. 88, pp. 10–28, 2017.

[8] X. Cao, D. M. Shila, Y. Cheng, Z. Yang, Y. Zhou, and J. Chen, "Ghost-in-zigbee: Energy depletion attack on zigbee-based wireless networks," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 816–829, 2016.

[9] P. Pongle and G. Chavan, "A survey: Attacks on rpl and 6lowpan in iot," in *2015 International Conference on Pervasive Computing (ICPC)*. IEEE, 2015, pp. 1–6.

[10] A. Mayzaud, R. Badonnel, and I. Chrisment, "A taxonomy of attacks in rpl-based internet of things," *International Journal of Network Security*, vol. 18, no. 3, pp. 459–473, 2016.

[11] K. Fu and W. Xu, "Risks of trusting the physics of sensors," *Communications of the ACM*, vol. 61, no. 2, pp. 20–23, 2018.

[12] Y. Shoukry, P. Martin, Y. Yona, S. Diggavi, and M. Srivastava, "Pycra: Physical challenge-response authentication for active sensors under spoofing attacks," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 1004–1015.

[13] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis *et al.*, "Understanding the mirai botnet," in *26th {USENIX} Security Symposium ({USENIX} Security 17)*, 2017, pp. 1093–1110.

[14] C. Li, Y. Wu, X. Yuan, Z. Sun, W. Wang, X. Li, and L. Gong, "Detection and defense of ddos attack–based on deep learning in openflow-based sdn," *International Journal of Communication Systems*, vol. 31, no. 5, p. e3497, 2018.

[15] M. N. Napiah, M. Y. I. B. Idris, R. Ramli, and I. Ahmedy, "Compression header analyzer intrusion detection system (cha-ids) for 6lowpan communication protocol," *IEEE Access*, vol. 6, pp. 16 623–16 638, 2018.

[16] D. Midi, A. Rullo, A. Mudgerikar, and E. Bertino, "Kalis—a system for knowledge-driven adaptable intrusion detection for the internet of things," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 656–666.

[17] T. D. Nguyen, S. Marchal, M. Miettinen, M. H. Dang, N. Asokan, and A. Sadeghi, "Dïot: A crowdsourced self-learning approach for detecting compromised iot devices," *CoRR*, vol. abs/1804.07474, 2018. [Online]. Available: http://arxiv.org/abs/1804.07474

[18] M. Miettinen, S. Marchal, I. Hafeez, N. Asokan, A.-R. Sadeghi, and S. Tarkoma, "Iot sentinel: Automated device-type identification for security enforcement in iot," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 2177–2184.

[19] W. Wang, M. Zhu, J. Wang, X. Zeng, and Z. Yang, "End-to-end encrypted traffic classification with one-dimensional convolution neural networks," in *2017 IEEE International Conference on Intelligence and Security Informatics (ISI)*. IEEE, 2017, pp. 43–48.

[20] Z. Wang, "The applications of deep learning on traffic identification," *BlackHat USA*, vol. 24, 2015.

[21] T. Luo, H.-P. Tan, and T. Q. Quek, "Sensor openflow: Enabling software-defined wireless sensor networks," *IEEE Communications letters*, vol. 16, no. 11, pp. 1896–1899, 2012.

[22] L. Galluccio, S. Milardo, G. Morabito, and S. Palazzo, "Sdn-wise: Design, prototyping and experimentation of a stateful sdn solution for wireless sensor networks," in *2015 IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2015, pp. 513–521.

[23] Y.-B. Lin, S.-Y. Wang, C.-C. Huang, and C.-M. Wu, "The sdn approach for the aggregation/disaggregation of sensor data," *Sensors*, vol. 18, no. 7, p. 2025, 2018.

[24] H. T. Dang, H. Wang, T. Jepsen, G. Brebner, C. Kim, J. Rexford, R. Soulé, and H. Weatherspoon, "Whippersnapper: A p4 language benchmark suite," in *Proceedings of the Symposium on SDN Research*. ACM, 2017, pp. 95–101.

[25] S. Haykin, *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994.

[26] A. Van Den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio." *SSW*, vol. 125, 2016.

[27] R. Yu, A. Li, C.-F. Chen, J.-H. Lai, V. I. Morariu, X. Han, M. Gao, C.-Y. Lin, and L. S. Davis, "Nisp: Pruning networks using neuron importance score propagation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 9194–9203.

[28] M. Verhelst and B. Moons, "Embedded deep neural network processing: Algorithmic and processor techniques bring deep learning to iot and edge devices," *IEEE Solid-State Circuits Magazine*, vol. 9, no. 4, pp. 55–65, 2017.

[29] G. Roffo, S. Melzi, and M. Cristani, "Infinite feature selection," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 4202–4210.

[30] D. P. Bertsekas, *Dynamic programming and optimal control*. Athena scientific Belmont, MA, 1995, vol. 1, no. 2.

[31] E. B. Beigi, H. H. Jazi, N. Stakhanova, and A. A. Ghorbani, "Towards effective feature selection in machine learning-based botnet detection approaches," in *2014 IEEE Conference on Communications and Network Security*. IEEE, 2014, pp. 247–255.

[32] A. H. Lashkari, A. F. A. Kadir, H. Gonzalez, K. F. Mbah, and A. A. Ghorbani, "Towards a network-based framework for android malware detection and characterization," in *2017 15th Annual Conference on Privacy, Security and Trust (PST)*. IEEE, 2017, pp. 233–23 309.

[33] A. Le, J. Loo, Y. Luo, and A. Lasebae, "The impacts of internal threats towards routing protocol for low power and lossy network performance," in *2013 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 2013, pp. 000 789–000 794.

[34] Libelium. (n.d.) Waspmote. [Online]. Available: http://www.libelium.com/products/waspmote/

[35] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ser. ICML'10. USA: Omnipress, 2010, pp. 807–814. [Online]. Available: http://dl.acm.org/citation.cfm?id=3104322.3104425

[36] S. Nanda, F. Zafari, C. DeCusatis, E. Wedaa, and B. Yang, "Predicting network attack patterns in sdn using machine learning approach," in *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. IEEE, 2016, pp. 167–172.

[37] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[38] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, 2016, pp. 265–283.

[39] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, ser. Hotnets-IX. New York, NY, USA: ACM, 2010, pp. 19:1–19:6. [Online]. Available: http://doi.acm.org/10.1145/1868447.1868466

[40] P. L. Consortium *et al.*, "Behavioral model (bmv2)," 2018.

[41] "Source codes and datasets," 2020. [Online]. Available: https://github.com/vxxx03/ICDCS2020

[42] P. Biondi *et al.*, "Scapy," 2011. [Online]. Available: https://scapy.net/