# Line-Speed and Scalable Intrusion Detection at the Network Edge via Federated Learning

Qiaofeng Qin[1], Konstantinos Poularakis[1], Kin K. Leung[2], and Leandros Tassiulas[1]
[1]Department of Electrical Engineering and Institute for Network Science, Yale University, USA
[2]Department of Electrical and Electronic Engineering, Imperial College London, UK

***Abstract*—Intrusion detection through classifying incoming packets is a crucial functionality at the network edge, requiring accuracy, efficiency and scalability at the same time, introducing a great challenge. On the one hand, traditional table-based switch functions have limited capacity to identify complicated network attack behaviors. On the other hand, machine learning based methods providing high accuracy are widely used for packet classification, but they typically require packets to be forwarded to an extra host and therefore increase the network latency. To overcome these limitations, in this paper we propose an architecture with programmable data plane switches. We show that Binarized Neural Networks (BNNs) can be implemented as switch functions at the network edge classifying incoming packets at the line speed of the switches. To train BNNs in a scalable manner, we adopt a federated learning approach that keeps the communication overheads of training small even for scenarios involving many edge network domains. We next develop a prototype using the P4 language and perform evaluations. The results demonstrate that a multi-fold improvement in latency and communication overheads can be achieved compared to state-of-the-art learning architectures.**

## I. Introduction

Edge networking attracts significant research interest with the rapid growth in the amount of mobile devices. Meanwhile, more security threats emerge in edge networking scenarios such as the botnet [1] where a hijacked edge device may infect more devices across different edge domains to conduct large-scale attacks. Therefore, it is necessary to deploy firewall functions and other security mechanisms at the network edge to identify harmful traffic flows from normal ones [2].

Machine learning algorithms such as neural networks are widely adopted for classifying incoming packets. Taking the values of packet header fields and flow statistics as input features, these algorithms are able to learn the pattern of attacks from collected network traces and make predictions for future inputs with high accuracy. However, traditional switches at the network edge only support relatively simple functions such as specific packet header fields matching and table lookup. Therefore, an unknown packet incoming to a switch has to be forwarded to a remote server or host where the learning algorithms run. The delay incurred makes it unlikely to process packets at a high speed. In addition, a large number of flow rules will be generated in this procedure and have to be stored in the switches, whose memory is usually limited and becomes another bottleneck [3].

The development of Software Defined Networking (SDN) and the programmable data plane concept in recent years bring new opportunities towards addressing the above challenges. SDN separates the control and data planes of a network, enabling an external network entity, known as the controller, to manage the data plane switches in a programmable manner. Furthermore, SmartNIC products and P4 language [4] enhance the capability of the switch itself, which is now capable of offloading services that are traditionally run in remote servers with general (and powerful) CPUs [5].

Binarized Neural Network (BNN) [6] can be used to deploy machine-learning-based packet classification in the form of in-network services inside the switches. BNN compresses all the weights of a neural network into single bits, therefore significantly reducing the computation and memory requirement of performing the inference to a level that a data plane switch may afford. It also converts all computations (e.g., real-valued dot production and activation functions) into bitwise operations, which are supported by typical programmable data plane switches.

While the use of BNNs can expedite the inference process by enabling the offloading of it directly on the data plane switch level, there still exist challenges about the training process of these learning models. It is unclear how to train the BNNs in a scalable manner e.g., in large networks with many interconnected edge domains, many gateways and switches. When a new attack pattern appears only in specific domains, other gateways should also be informed, even if the attacker's packets do not go through them, so as to make more efficient training decisions in future. Meanwhile, the communication overheads either among gateways or between the gateways and the cloud being responsible for the training should also be considered. Even worse, it is possible that edge domains are controlled by multiple parties who do not want to share their network traces with others for training, since the information leak itself is another security threat.

Federated learning [7] is a technique suitable for online training in this scenario, which aggregates local weight updates from each gateway without asking their collected packets, and then calculates new model parameters for gateways. We explore a novel way of combining federated learning and BNN to set up a scalable packet classification architecture with high performance and low costs while preserving the privacy of

ISBN 978-3-903176-28-7© 2020 IFIP
This publication was supported partly by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001, the National Science Foundation under Grant CNS 1815676, and the Office of Naval Research under Grant N00014-19-1-2566.

network traces.

Specifically, we make the following contributions:

- We propose a learning framework for packet classification combining BNNs and federated learning achieving high accuracy with low memory and communication costs. To the best of our knowledge, this is the first work combining these concepts together.
- We design an architecture based on programmable network switches for providing security service to multi-party edge device owners while performing packet classification at the line speed of the switches and updating learning models in a scalable manner.
- We develop a prototype of the proposed architecture in P4 language and evaluate its performance and costs in a network testbed with real devices and traffic traces. We find that a multi-fold improvement in latency and communication overheads can be achieved compared to state-of-the-art learning architectures.

The remainder of this paper is organized as follows. After discussing our contribution over related works in Section II, we describe the main challenges of the packet classification at the network edge and propose a system architecture in Section III. In Section IV, we describe the learning model inference and training mechanisms, as well as the federated learning framework. Section V demonstrates how such architecture and mechanisms are implemented as a prototype, and Section VI evaluates its performance. We conclude the paper in Section VII.

## II. Related Work

**SDN and In-network Processing**. Software Defined Networking (SDN) provides programmable and centralized network management by separating the control and data planes. By placing multiple controllers in different domains, SDN can scale well in a multi-domain edge network scenario [8]. As for the data plane, a trend is to make switches programmable, such as the development of P4 language [4]. P4 enables in-network processing by deploying services in switches instead of servers. [5] investigates various in-network processing applications which show high efficiency and lower costs compared to traditional methods. [9] adopts such approach at the network edge.

**Learning Methods**. Machine learning has been widely used for packet classification and intrusion detection such as approaches in [10] [11] promising high accuracy. However, a remote host or server is typically required to run the learning algorithm, introducing additional latency and preventing packets from being processed at the line speed of the switches. This is true even for the SDN-based learning methods [12] where learning is performed in the control plane (SDN controller) and the data plane (switches) only plays the role of flow table storing and matching. To overcome this limitation, we seek for a data plane-compatible algorithm for higher processing speed.

**Binarized Neural Networks**. BNN is a type of neural network with only binary weights and activation functions [6], the inference process of which can be converted into bitwise operations. [13] demonstrates that BNN can achieve much faster speed and cost less memory while maintaining a high level of accuracy. Such features make it suitable for embedded devices with limited capacity [14]. [15] and [16] attempted to implement BNN in smart network devices. We make similar attempts while also performing realistic networking tasks, i.e., packet classification. In addition, we propose an online training scheme, which is scalable by adopting federated learning techniques.

**Distributed / Federated Learning**. For better scalability, neural networks can be trained in a distributed manner. Furthermore, the concept of federated learning is proposed [7], which keeps the training data locally to preserve privacy. Federated learning has been applied for the security issue in edge scenarios, e.g., IoT [17] and mobile networks [18]. Reducing communication overhead is a major concern in distributed and federated learning. One promising approach is to quantize or binarize the weight updates, such as SignSGD [19]. The distributed learning procedure also shows good compatibility with programmable data plane devices. [20] and [21] propose in-network methods for accelerating the aggregation phase of distributed training. In this paper, we explore methods for effective intrusion detection at the network edge by combining the advantages of federated learning, BNN and programmable data plane.

## III. System Architecture

In this section, we describe the architecture design of the proposed system for network security. The system consists of a central cloud and several edge network domains. For each domain, there is a gateway node responsible for forwarding packets from and to the devices of that domain. It also performs packet classification to identify attacks from normal traffic flows. Each gateway is SDN-enabled with separated control and data plane i.e., an edge controller and a switch. Both planes are programmable. Previous works have shown the feasibility and benefits of this type of gateway design and implementation for edge networking scenarios [9]. In this work, we make a step further and propose specific mechanisms for effective packet classification achieving high accuracy with low memory and communication costs. We first list a number of challenges we need to address before presenting the proposed mechanisms.

### A. Challenges

A high-performance architecture for packet classification at the network edge has multiple requirements:

1) **High Accuracy & Low False Alarm Rate**. The gateway should be capable to identify attacks from normal flows. Besides, the false alarms (normal packets incorrectly classified as attack packets) must be kept to a low rate, otherwise normal packets may be blocked and network functions will be hampered.
2) **Line-Speed Packet Processing**. The gateway should perform the packet classification by itself instead of

forwarding packets to a remote host or server and waiting for reply. This requires the classification algorithm (inference process of the learning algorithm) to be lightweight enough so that the gateway can run it locally in real time.

3) **Model Updates**. An edge domain can be highly dynamic with new devices joining the network and new traffic flows generated over time. The gateway should be able to use the new network traces to improve the classification algorithm, i.e., re-train the model over time. The training task can be offloaded to the control plane or remote cloud server, but the updated model must be finally downloaded to the gateway data plane.
4) **Scalability and Privacy**. It is common in an edge networking scenario that the amount of devices and domains is large. A solution can hardly scale up unless the communication overheads between the cloud server and gateways during training can be controlled in a reasonable manner. In addition, devices of different edge domains may belong to different owners who are not willing to share their network traces for training.

### B. Design Choices

In order to meet all requirements above, we choose the binarized neural network (BNN) and federated learning as the main components of our architecture. We describe each component in the following, as depicted in Figure 1.

**Gateway Data Plane (Programmable Switches)**. The data plane refers to a packet forwarding device with programmability such as P4-enabled switches, SmartNICs and FPGAs. A BNN is deployed in each gateway's data plane for classifying incoming packets. The data plane extracts certain bits from incoming packet's header as the BNN input and a binary output (i.e., attack or normal traffic) is acquired by a series of bitwise operations. After this inference process, the gateway performs ordinary packet forwarding for normal traffic and is able to send attack samples to the control plane if online training is active. *With both the classification and forwarding functions inside the data plane, line-speed packet processing can be achieved.*

**Gateway Control Plane (Edge Controllers)**. Each gateway is managed by a separate edge controller with a general CPU or GPU. The controller may be deployed locally in the gateway or in another host within the same domain. The controller maintains a neural network with the same structure as in the data plane, except that the weights and activation functions are not binarized. This neural network is used for re-training the classification algorithm over time by performing backward propagation with the new network traces collected by the data plane. The controller also keeps an API writing weight values to the data plane, and an API communicating with the cloud server for federated learning. The detailed methods will be introduced in the next section.

**Cloud Server**. For scalable training of the classification algorithm, a federated learning technique [7] is deployed in the cloud server. The federated learning can be regarded as a service provided by the cloud, and each gateway can choose whether to subscribe to this service, decided by its owner. Each gateway subscribing to the service, after each epoch of local training, it sends the local updates to the cloud that acts as the aggregator. When the aggregator receives messages from all the gateways, it calculates the new model weights based on the local updates and broadcasts the new model weights to the gateways.

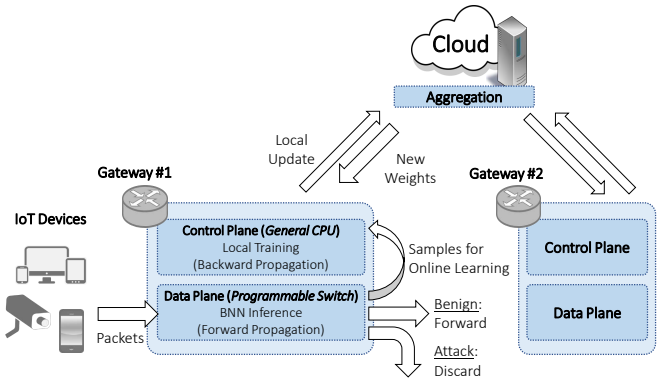



Fig. 1. An architecture deploying BNN and federated learning for network security at the edge.

The procedures of BNN inference in the data plane, model training in the control plane and weight aggregation in the cloud as well as the implementation details of these mechanisms will be described in the next sections.

## IV. Problem Definition & Solution

In this section, we formally define the packet classification problem at the network edge and describe how we adopt BNN and federated learning techniques to solve it.

### A. Problem Formulation

We consider a system of a cloud server $c$ and $N$ edge network domains. Each domain contains a gateway which is the pair of a data plane switch and its edge controller (collocated with the switch or hosted in a different device within the same domain). The set of all gateways is denoted by $\mathbf{N}$.

A data plane switch is able to parse headers of different protocols contained in a packet and determine where the packet should be forwarded (or blocked) according to specific header fields, which can be regarded as *packet-level features*. The switch may also use *flow-level features* such as the packet/byte count of a flow to make appropriate forwarding decisions. It is straightforward to represent both types of features by a bit string. Therefore, given a group of features supported by the gateway, we can concatenate them with a fixed sequence to get a 1D vector. Each element of the vector is binary, i.e., either $-1$ or $+1$. We denote this vector as $x_0$, which is the input for the packet classification.

The purpose of packet classification is to find a function $\hat{y} = f_n(x_0)$ at each gateway $n \in \mathbf{N}$, where $\hat{y}$ is a 1D binary vector indicating the prediction of the packet type. For example, as

**Algorithm 1** Inference Process

**Input:**
$x_0$: binary input sample
$W^b_{n,l}$: binary weights of layer $l$ in gateway $n$'s data plane
**Output:**
$y$: binary prediction

**for** $l = 1 : L-1$ **do**
    $x_l \leftarrow sign(XnorDotProduct(x_{l-1}, W^b_{n,l}))$
**end for**
$y \leftarrow sign(XnorDotProduct(x_{L-1}, W^b_{n,L}))$

a simple case, $\hat{y}$ has only one binary element, taking value of $+1$ if the incoming packet belongs to a normal traffic flow, or $-1$ if it belongs to an attack.

### *B. Inference: Binarized Neural Networks*

To achieve line-speed packet processing, we require that an incoming packet is classified directly in the gateway instead of forwarded to the edge controller or any other remote server. In other words, each gateway $n$ executes $f_n(x_0)$ in its data plane independently without help from either its edge controller or gateways of other domains.

Neural network is one of the most popular methods for packet classification. However, it requires a large amount of dot product operations on real-valued vectors, as well as activation functions which are usually non-linear. Originally designed for packet forwarding, most data plane devices do not support these operations. To overcome this difficulty, we deploy BNN [6] that has weights of only binary ($+1$ or $-1$) values and sign function as the activation function. More specifically, consider a neural network with $L$ fully-connected layers. We denote the neuron weights of layer $l$ by a 2D vector $W^b_{n,l}$ and denote the input of this layer by $x_{l-1}$. Then, the output of layer $l$ is:

$$x_l = sign(x_{l-1} \cdot W^b_{n,l}) \tag{1}$$

If both $x_{l-1}$ and $W^b_{n,l}$ are binary vectors, this operation is equivalent to the Hamming weight of two bit strings' XNOR. Similarly, the whole inference procedure of $L$ layers is described in Algorithm 1. In the next section, we will demonstrate how we implement it completely in a programmable data plane device.

### *C. Training: Federated Learning Technique*

To classify packets with high accuracy, a neural network needs to be trained in order to get optimal weights. Although BNN is efficient when performing the inference, it cannot be trained directly because gradients cannot be calculated from binary functions. We adopt a similar method as [6], which keeps the real-valued weights denoted by $W_n$. When calculating the loss function by forward propagation, binary weights are used. However, during the backward propagation as the next step, real-valued gradients are calculated and applied for the weight update. In our approach, we store $W_n$ and perform the backward propagation in the *edge controller*

**Algorithm 2** Training Process

**Input:**
$X_n, Y_n$: batch of inputs and labels trained at gateway $n$
$L(\hat{Y}_n, Y_n)$: loss function
$W^t_n$: real-valued weights in gateway $n$'s control plane
$W^{b,t}_n$: binary weights in gateway $n$'s data plane
$\delta^t$: learning rate
**Output:**
$W^{t+1}_n$,$W^{b,t+1}_n$: updated weights of each gateway

**for** $n \in \mathbf{N}$ **do**
    $\hat{Y}_n \leftarrow ForwardPropogation(X, W^t_n, W^{b,t}_n)$
    $g_n \leftarrow BackPropogation(L(\hat{Y}_n, Y), W^t_n)$
**end for**
(At the cloud) $\Delta W \leftarrow \delta^t sign[\sum_{n=1}^{N} sign(g_n)]$
**for** $n \in \mathbf{N}$ **do**
    $W^{t+1}_n \leftarrow W^t_n + \Delta W$
    $W^{b,t+1}_n \leftarrow sign(W^{t+1}_n)$
**end for**

of the gateway $n$, leaving the data plane for binary forward propagation only. Besides this one-time training, it is also possible for the data plane to report the inference results of incoming packets to its controller in real time, so that training can be performed again over time in the controller to improve the classification accuracy.

[13] suggests that replacing the output layer with real-valued weights and activation functions during the forward propagation will positively impact the accuracy in practice. Such improvement is also possible in our architecture. The data plane can send to the controller the output bit string of its BNN's last hidden layer and make the controller finish the calculation of the output layer using the real-valued weights. The details of the interaction between control and data planes will be described in the next section.

So far, we have discussed the BNN training within one edge domain. In a network with $N$ domains, each domain's gateway may receive different packet samples. In order to learn more comprehensive attack patterns, we adopt federated learning [7] across all domains by connecting all gateway controllers to a cloud server. In federated learning, each gateway calculates the weight gradients with a batch of local input samples and sends the local updates to the cloud. Receiving updates from all gateways, the cloud will aggregate them and announce new weight values.

Scalability of federated learning is one of our main concerns. With a large number $N$ of domains, the communication overheads between controllers and the cloud are not negligible if each controller reports all its real-valued weight updates in every learning batch. To save bandwidth, we take another binarization approach, SignSGD [19]. According to this method, each gateway now reports the 1-bit sign of local updates. Then, the cloud will have a "majority vote" and announce the result, which are also single bits. More specifically, we denote a local update of gateway $n$ by $g_n$, then the new weights after

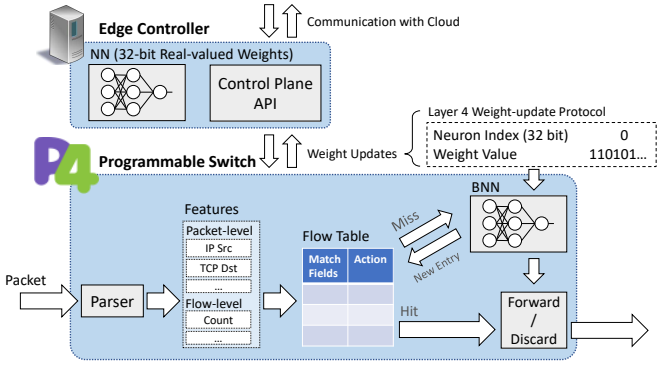



Fig. 2. P4-based prototype of the proposed gateway in one domain.

communicating with the cloud are calculated by:

$$W_n^{t+1} = W_n^t + \delta^t sign[\sum_{n=1}^{N} sign(g_n)] \quad (2)$$

where $\delta^t$ is the learning rate. Both down-link and up-link messages during federated learning are compressed to single bits, while the convergence persists as proven in [19]. The complete BNN federated learning process is described in Algorithm 2.

Intuitively, SignSGD is expected to cooperate well with BNN because $W_n^b$ will not change unless the update to $W_n$ is large enough, i.e., from a negative value to a positive one or the other way around. Updates without impact on $W_n^b$ will become a waste of resources. On the other hand, (2) appears to be a suitable way of updating. We will further show the efficiency of this proposed method in the evaluation section.

## V. Implementation

In this section, we develop a prototype of the proposed architecture. Among various available programmable data plane methods, we choose the representative P4 language [4] to implement our system. P4 is capable of achieving relatively complicated logic of packet header parsing and stateful processing, and it can be compiled for various targets, i.e., different types of software/hardware switches.

### A. P4 Data Plane

The data plane device (gateway) in each domain runs a P4 program which is the key component of our proposed architecture. It is responsible for the following functionalities.

**Feature Extraction.** Protocol-independence is one of the most significant features of P4. By defining different network protocol headers in a P4 program, the data plane device is able to extract any header fields (e.g., fields of IP, TCP and even application layer protocols like HTTP) from an incoming packet and interpret them as bit strings. We concatenate several such strings together as the input of the BNN. Moreover, P4 also provides multiple ways (e.g., meters, counters and registers) to extract flow-level statistics. Such features can be used as the input of the BNN in the same way.

```
// an example of 120-bit input and 120 neurons in each layer
control MyIngress(...) {
  register<bit<120>>(1024) weights;
  bit<120> Input = 0;
  bit<120> NextLayerInput = 0;
  bit<1> Activated;

  action Activation(bit<120> NeuronInput){
    bit<8> popcnt = ...  // calculate Hamming weight
    Activated = popcnt>60;
    NextLayerInput = NextLayerInput<<1 + (bit<120>) Activated;
  }

  action LayerProcess(bit<10> IndexOffset){
    bit<120> weight = 0;
    weights.read(weight, (bit<32>)IndexOffset+0);
    Activation(~(weight^Input));
    weights.read(weight, (bit<32>)IndexOffset+1);
    Activation(~(weight^Input));
    ...  // process all neurons in the same way
    }

  apply{
    ...
    // a function extracting header fields and statistics
    BuildInput();
    LayerProcess(0);        // first layer processing
    Input=NextLayerInput;
    NextLayerInput=0;
    LayerProcess(120);      // second layer processing
    Input=NextLayerInput;
    NextLayerInput=0;
    LayerProcess(240);      // third layer processing
    ...
  }
  ...
}
```

Fig. 3. Implementing BNN with P4 codes

**BNN Implementation.** We use a register to store each BNN neuron's weight as a bit string. The registers are stateful so that they can be written and read dynamically. When processing each layer, bitwise XNOR operations are performed between the input bit string and every neuron in the layer. The activation function can be realized by calculating the Hamming weight of the XNOR output. Although P4 does not provide built-in functions for it, there are various works [22] providing algorithms that enable fast calculations, and the parallel algorithm among them can be easily implemented in P4. Figure 3 roughly shows how BNN can be implemented using the P4 grammar and data structure. In addition, we also implement the same logic in C language for supported devices.

**Packet Forwarding.** The BNN can coexist with layer-2/3 or any custom packet forwarding mechanism in the same P4 program. In this prototype, we consider a simple case where a packet from the flow regarded as an attack will be directly discarded. We combine the BNN with a flow table matching the incoming packet's 5-tuple. If the packet hits an entry in the flow table, it will be processed accordingly without being sent to the BNN. Otherwise, the BNN performs inference and adds a new entry to the table. *In both cases, line-speed processing is achieved, and this method further improves the efficiency as well as reduces computation costs.* The whole workflow of the data plane is depicted in Figure 2.

**Control Protocol.** We define a new layer-4 protocol for the control plane to update the weight values of the data plane. It

```
typedef bit<120> MaxInputSize;
header weightupdate_t {
    bit<32>             index;
    MaxInputSize        value;
}
...
parser MyParser(...){
    ...
    state parse_ipv4 {
        packet.extract(hdr.ipv4);
        transition select(hdr.ipv4.protocol) {
            17: parse_udp;
            6:  parse_tcp;
            61: parse_weightupdate;
            default: accept;
        }
    }

    state parse_weightupdate {
        packet.extract(hdr.weightupdate);
        transition accept;
    }
}
```

Fig. 4. An example of P4 header definition for weight updates

contains two header fields as shown in Figure 4, the index of target neuron and a bit string representing the weight values of this neuron. When the data plane device receives a packet carrying this header from the controller, it will neither forward this packet nor call the BNN. Instead, it writes the new weight value to its register. This protocol can also be used by the data plane to send the output of the BNN's last hidden layer to the controller during the online training process, as described in the previous section.

### B. Control Plane and Cloud Server

We deploy another host with a general CPU in the same domain as the controller for each gateway. In order to perform online training, each controller should hold a neural network with real-valued (rather than binary) weights. We implement such networks by TensorFlow [23] and use Scapy [24] for the communications with the data plane. We also deploy a server as the cloud for federated learning. It receives local updates from each controller through UDP packets and conducts the aggregation. We evaluate this prototype with different topologies, which will be described in detail in the next section.

## VI. Evaluation

In this section, we deploy the proposed architecture and algorithms in a network testbed and evaluate them with a mixture of emulations and real device experiments to demonstrate the performance and costs in multiple aspects.

### A. Testbed Setup

We set up a network testbed containing multiple desktop computers with Linux operating system, connected through Ethernet cables. Each domain as well as the cloud server is represented by one computer. Each domain contains multiple hosts and one gateway, which are deployed in a Mininet [25] virtual network. We compile the data plane P4 program to BMv2 [26] software switches. The BNN implemented inside the data plane contains one fully-connected hidden layer with 120 neurons and a single-neuron output.

We consider the following publicly available datasets containing network traces to train and test the packet classification algorithm.

- **CICIDS2017** [27]. This dataset has a labeled record of multiple types of attacks and benign flows. Statistics are summarized for each flow. We take two thirds of records for training and the remaining for testing. We convert the layer-4 destination port, bidirectional total amount of packets and bytes into a 144-bit input vector to the BNN. All these statistics can be easily acquired by a P4-enabled switch.
- **ISCX Botnet 2014** [28]. This dataset collects heterogeneous botnet and malware traffic in realistic scenarios as well as non-malicious traffic. Its test set contains larger diversity than the training set to evaluate whether an algorithm is able to handle unknown traffic patterns. For the evaluations, we replay the TCP and UDP flows in this dataset to the gateway. Different from the last dataset, we choose a very common group of packet-level features, 5-tuple (IP addresses, layer-4 protocol and ports) and IP packet length as a 120-bit input vector.

### B. Performance of Inference

First, we concentrate on Algorithm 1 and evaluate the classification performance within the scope of one domain and one gateway. Ignoring the federated learning method temporarily, we conduct an offline training on the gateway's BNN with the complete dataset and Adam [29] optimizer. For comparison, we also adopt other state-of-the-art learning algorithms, including the decision tree (DT) and linear support-vector machine (SVM) methods implemented by scikit-learn [30], as well as another neural network (denoted by NN) having the same structure as our BNN except that the activation function is non-linear (sigmoid function) and all weights are real-valued with 32-bit precision. Comparison with this NN will indicate if the binarization leads to performance loss.

We measure multiple metrics characterizing the performance of inference, calculated as follows:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3}$$

$$precision = \frac{TP}{TP + FP}, \quad recall = \frac{TP}{TP + FN} \tag{4}$$

where $TP$, $FP$, $FN$, $TN$ are abbreviations denoting the amount of true positives, false positives, false negatives, and true negatives. We finally calculate the F-1 score defined as the harmonic mean of precision and recall:

$$F_1 = 2 * \frac{precision * recall}{precision + recall} \tag{5}$$

**Flow-Level Classification**. Table I contains our measurement of accuracy, precision and recall rates on CICIDS2017 dataset, where algorithms classify a flow based on several statistics. We observe that the real-valued NN has the same

level of performance with DT. Our proposed BNN method has only slightly lower accuracy ($0.6\%$) after the binarization. It also behaves better than SVM. At the same time, the BNN compresses the memory required for weight value storage to $1/32$ compared with the real-valued NN and makes it possible to run the algorithm as a data plane switch function (at the line speed of the switches). Besides, although DT has a good performance here, it lacks an effectively training algorithm in a distributed manner [18]. In contrast, we will demonstrate how the BNN can be trained across different domains using the federated learning framework in the next subsection.

| Method | Accuracy | Precision | Recall | $F_1$ |
|---|---|---|---|---|
| **BNN** | **0.983** | 0.966 | 0.963 | **0.965** |
| NN | 0.989 | 0.967 | 0.987 | 0.977 |
| DT | 0.989 | 0.962 | 0.993 | 0.977 |
| SVM | 0.957 | 0.889 | 0.937 | 0.913 |

TABLE I
PERFORMANCE METRICS ON CICIDS2017 DATASET.

**Packet-Level Classification**. While we have shown that our method is valid when performing classification based on flow statistics, we now concentrate on the packet-level features, i.e., matching on header fields, which permits the switch to react to incoming packets in real time. This is the major use case of the proposed method as a switch function. We measure performance metrics on the Botnet 2014 dataset with such packet-level features as inputs in Table II. As in the previous table, we observe that the binarization incurs minor accuracy loss only ($1.05\%$). Besides, BNN behaves better than both DT and SVM ($6\%$ and $7\%$ more accuracy) under this setting.

| Method | Accuracy | Precision | Recall | $F_1$ |
|---|---|---|---|---|
| **BNN** | **0.945** | 0.945 | 0.766 | **0.846** |
| NN | 0.953 | 0.992 | 0.767 | 0.865 |
| DT | 0.900 | 0.735 | 0.767 | 0.751 |
| SVM | 0.890 | 0.700 | 0.763 | 0.730 |

TABLE II
PERFORMANCE METRICS ON BOTNET 2014 DATASET.

A high recall rate is especially important for packet classification, since the incorrect blockage of non-malicious traffic (false negatives) may hamper normal network functionalities. Therefore we also measure the precision and recall rates in Table II and calculate the F-1 score, which shows a similar tendency as the accuracy performance.

Moreover, by adjusting the threshold of the Hamming weight calculated in the output layer, a tradeoff can be achieved as depicted in Figure 5, which means that a better (higher) recall rate can be acquired at a cost of sacrificing some precision.

**Packet Processing Latency**. We next examine how the line-speed packet classification can be achieved in our proposed architecture. We send a subset of the Botnet 2014 dataset containing 2000 successive packets from a host to the gateway. As described in last section, the gateway data plane (the programmable switch) keeps both the BNN and a flow table

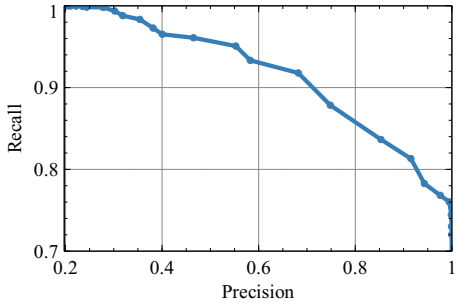
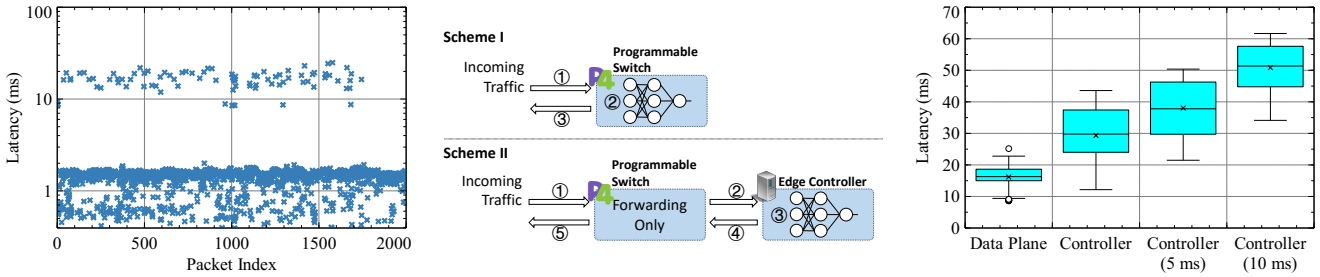



Fig. 5. The precision-recall curve.

matching the source IP addresses and TCP/UDP ports of incoming packets. In order to measure the network latency of every packet correctly, the switch marks the packets of malicious flows in the DSCP field instead of dropping them. Figure 6(a) plots the distribution of network latency of each packet. A small portion (around $5\%$) of packets are processed with a larger latency, having an order of magnitude of $10$ ms. These are unknown input samples the gateway encounters for the first time without having a table entry, and therefore the switch uses the BNN to process them. The remaining $95\%$ packets are processed with a much smaller latency (less than $2$ ms), because they just require a one-time flow table match operation.

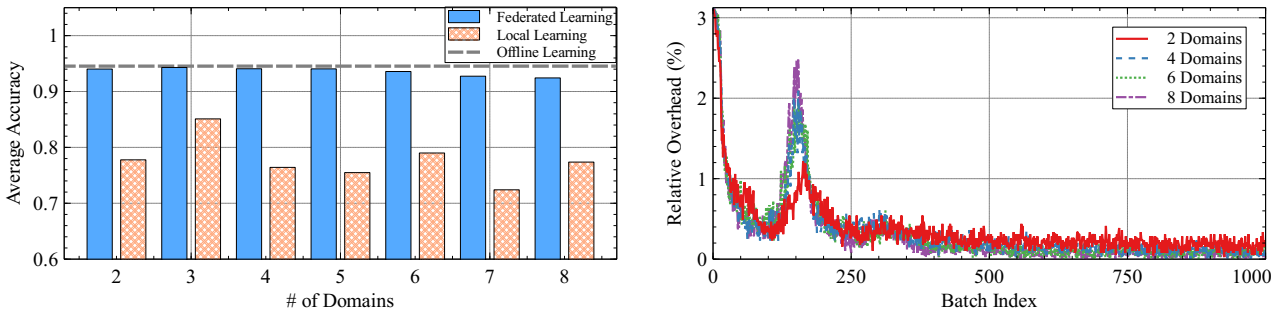
We next focus on the latency caused by running BNN in the control plane, which involves more complicated calculations. We deploy an alternative architecture (Scheme II in Figure 6(b)) where the neural network is deployed in the edge controller within the same domain. In this case, the data plane switch has to forward an unknown packet to the controller before making forwarding decisions. This is similar to the traditional intrusion detection approaches. To evaluate the performance of the two different architectures, we disable the flow table and make the BNN to process all packets. The box plots of latency are depicted in Figure 6(c). We notice that both the average value and the variation of packet processing latency are lower when deploying the BNN directly in the data plane. Moreover, unlike the emulation environment, there is usually also propagation delay between the data and control planes in reality. Therefore, we introduce extra delay at the link of the control path (the third and forth box plots). As a result, the packet processing latency increases accordingly, demonstrating further the efficiency of our programmable data plane approach.

**Hardware Support**. The BMv2 software switch is not designed for production-grade performance. Therefore, we also deploy a Netronome Agilio CX SmartNIC with $10$ GbE ports. It is programmable by supporting a mixture of P4 and C codes, permitting us to further optimize the proposed functionality by implementing the header parsing in P4 and BNN in C to take full advantage of this high-performance device. We will deploy such hardware in a larger scale to have more realistic evaluations as a future work. We also make




(a) Latency distribution. (b) Different architectures. (c) Latency of processing unknown samples.

Fig. 6. Packet processing latency evaluations of BNN inference as a switch function in the data plane.




(a) Accuracy of different learning schemes. (b) Convergence and control message overhead.

Fig. 7. The (a) accuracy and (b) control message overheads during federated learning with the network scaling.

both our BMv2 and Agilio codes publicly available [31] for the benefit of the research community.

**Main Takeaways.** (1) The proposed BNN method performs packet classification with high accuracy based on both flow-level (flow statistics) and packet-level (header fields) features. (2) The BNN method outperforms several state-of-the-art learning methods in accuracy and F-1 score, with only slight performance loss during the binarization. (3) Implementing BNN in the data plane as a switch function achieves faster packet processing speed (line speed) than traditional approaches that deploy similar functions in a remote host.

### C. Performance of Federated Learning

Having shown the performance of the proposed architecture within a single domain, we now extend the scenario to a multi-domain network and evaluate the federated learning method (Algorithm 2). We assume that there are $N$ domains each containing a gateway with the same P4 program. Correspondingly, the dataset is split into $N$ subsets, and each gateway can only get access to one of them.

**Accuracy with Distributed Training**. First, we consider a case without federated learning (denoted as local learning), where each gateway does not connect to the cloud and is trained based on its subset only. We evaluate the trained BNN in each domain's gateway with the original test set. The average accuracy is depicted by red bars with cross texture in Figure 7(a), which severely degrades (less than $80\%$ in the worst case compared with $94.5\%$ when training with the complete dataset). On the other hand, if the federated learning described in Algotihm 2 is adopted during training, we can get an accuracy (blue bars in Figure 7(a)) which is almost as good as the offline trainig with the complete dataset. Such conclusion holds with different $N$ values.

**Communication Overhead**. Although federated learning makes it possible to have a scalable solution for training gateways in multiple domains, the communication overhead of both uploading (gateways sending local updates) and downloading (the cloud announcing the aggregated update) will be a problem, especially when there is a large amount of domains, which is the reason why we apply the binarization technique the second time during this communication. We analyze two types of traffic overheads; between the cloud and gateway controllers, as well as between each gateway's control and data planes.

When analyzing the overheads, we compare with traditional federated learning approaches, where local updates are updated with real values usually represented by 32 bits. Then, the cloud will aggregate updates by calculating the average values. It will broadcast the aggregated weight updates also in 32 bits. It is straightforward that the SignSGD method we adopt will significantly reduce the traffic overheads between the cloud and each edge controller, because only a single bit for every weight is required in our approach, leading to $1/32$ up-link traffic overhead. The same analysis can also be applied for down-link overhead.

The control message overhead from a gateway controller to the data plane switch updating the binarized neural weights also decreases. Another benefit of replacing the real-valued

weights with single bits is that the controller does not need to send a control message if all binarized weights of the same neuron remain unchanged after training with a new batch. Therefore, less messages and overheads are required when the BNN converges. In Figure 7(b), we plot the control message overhead between all pairs of control and data planes during the first one thousand batches of federated learning. With the network converging quickly after training with $500$ batches, the overhead reduces to less than $0.5\%$ compared with the case that we use the real-valued NN and traditional federated learning method.

**Main Takeaways.** The proposed architecture enabled by federated learning leads to (1) much more accurate classification compared with training each gateway independently, and (2) small traffic overheads in communications between the cloud and edge controllers, as well as between the control and data planes.

## VII. Conclusion

In this paper, we explored new methods for enhancing security at the network edge with SDN and programmable data plane. We designed an architecture running BNNs in edge gateways as switch functions to detect attacks from incoming packets. We also proposed a federated learning framework for gateways of multiple edge network domains to learn new attack patterns online and collaboratively. Evaluations on a real prototype we developed demonstrate that our method can achieve line-speed packet processing with high classification accuracy and low false alarm rate. Moreover, our solution is scalable with small communication overheads between the control and data planes of each edge domain, as well as between the cloud and each edge controller.

## 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] T. Markham and C. Payne, "Security at the network edge: A distributed firewall architecture," in *Proceedings DARPA Information Survivability Conference and Exposition II. DISCEX'01*, vol. 1. IEEE, 2001, pp. 279–286.

[3] M. Kuźniar, P. Perešíni, and D. Kostić, "What you need to know about sdn flow tables," in *International Conference on Passive and Active Network Measurement*. Springer, 2015, pp. 347–359.

[4] 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.

[5] Y. Tokusashi, H. T. Dang, F. Pedone, R. Soulé, and N. Zilberman, "The case for in-network computing on demand," in *Proceedings of the Fourteenth EuroSys Conference 2019*, ser. EuroSys '19. New York, NY, USA: ACM, 2019, pp. 21:1–21:16. [Online]. Available: http://doi.acm.org/10.1145/3302424.3303979

[6] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1," *arXiv preprint arXiv:1602.02830*, 2016.

[7] B. McMahan and D. Ramage, "Federated learning: Collaborative machine learning without centralized training data," *Google Research Blog*, vol. 3, 2017.

[8] Q. Qin, K. Poularakis, G. Iosifidis, and L. Tassiulas, "Sdn controller placement at the edge: Optimizing delay and overheads," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, April 2018, pp. 684–692.

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

[10] 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.

[11] 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.

[12] P. Amaral, J. Dinis, P. Pinto, L. Bernardo, J. Tavares, and H. S. Mamede, "Machine learning in software defined networks: Data collection and traffic classification," in *2016 IEEE 24th International Conference on Network Protocols (ICNP)*. IEEE, 2016, pp. 1–5.

[13] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European Conference on Computer Vision*. Springer, 2016, pp. 525–542.

[14] B. McDanel, S. Teerapittayanon, and H. Kung, "Embedded binarized neural networks," *arXiv preprint arXiv:1709.02260*, 2017.

[15] G. Siracusano and R. Bifulco, "In-network neural networks," *arXiv preprint arXiv:1801.05731*, 2018.

[16] G. Siracusano, D. Sanvito, S. Galea, and R. Bifulco, "Deep learning inference on commodity network interface cards."

[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] E. Bakopoulou, B. Tillman, and A. Markopoulou, "A federated learning approach for mobile packet classification," *arXiv preprint arXiv:1907.13113*, 2019.

[19] J. Bernstein, Y.-X. Wang, K. Azizzadenesheli, and A. Anandkumar, "signsgd: Compressed optimisation for non-convex problems," *arXiv preprint arXiv:1802.04434*, 2018.

[20] A. Sapio, M. Canini, C.-Y. Ho, J. Nelson, P. Kalnis, C. Kim, A. Krishnamurthy, M. Moshref, D. R. Ports, and P. Richtárik, "Scaling distributed machine learning with in-network aggregation," *arXiv preprint arXiv:1903.06701*, 2019.

[21] Y. Li, I.-J. Liu, Y. Yuan, D. Chen, A. Schwing, and J. Huang, "Accelerating distributed reinforcement learning with in-switch computing," 2019.

[22] W. Muła, N. Kurz, and D. Lemire, "Faster population counts using avx2 instructions," *The Computer Journal*, vol. 61, no. 1, pp. 111–120, 2017.

[23] 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.

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

[25] 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

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

[27] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization." in *ICISSP*, 2018, pp. 108–116.

[28] 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.

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

[30] 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.

[31] "Source codes," 2020. [Online]. Available: https://github.com/vxxx03/IFIPNetworking20