2022 18th International Conference on Mobility, Sensing and Networking (MSN) | 978-1-6654-6457-4/22/$31.00 ©2022 IEEE | DOI: 10.1109/MSN57253.2022.00155

Binary Neural Network with P4 on Programmable Data Plane

Junming Luo
School of Electronics and Communication Engineering
Guangzhou University
Guangzhou, China, 510006
luojunming556@163.com

Waixi Liu
School of Electronics and Communication Engineering
Guangzhou University
Guangzhou, China, 510006
lwx@gzhu.edu.cn

Miaoquan Tan
School of Electronics and Communication Engineering
Guangzhou University
Guangzhou, China, 510006
tmqdyx24@163.com

Haosen Chen
School of Electronics and Communication Engineering
Guangzhou University
Guangzhou, China, 510006
csen_y@163.com

Abstract—**Deploying machine learning (ML) on the programmable data plane (PDP) has some unique advantages, such as quickly responding to network dynamics. However, compared to demands of ML, PDP have limited operations, computing and memory resources. Thus, some works only deploy simple traditional ML approaches (e.g., decision tree, K-means) on PDP, but their performance is not satisfactory. In this article, we propose P4-BNN (Binary Neural Network based on P4), which uses P4 to completely executes binary neural network on PDP. P4-BNN addresses some challenges. First, in order to use shift and simple integer arithmetic operations to replace multiplication, P4-BNN proposes a tailor-made data structure. Second, we use an equivalent replacement programming method to support matrix operation required by ML. Third, we propose a normalization method in PDP which needn't floating-point operations. Fourth, by using register storing the model parameters, the weights of P4-BNN model can be updated without interrupting the P4 program running. Finally, as two use-cases, we deploy P4-BNN on a Netronome SmartNIC (Agilio CX 2x10GbE) to achieve flow classification and anomaly detection. Compared to the N3IC, decision tree and K-means, the accuracy of P4-BNN has 1.7%, 3.4% and 47.7% improvement respectively.**

Keywords—binary neural network; P4; Programmable Data Plane; anomaly detection; flow classification

I. Introduction

In recent years, we have seen an increasing interest in deploying machine learning (ML) to solve network problems [1]. The software-defined networking (SDN) with flexible programming has also promoted the deploying of ML. However, deploying ML on control plane of SDN has some shortcomings: the long latency of control loop, and additional overhead required to obtain global view. For example, in the case of flow classification, FlowSeer [2] generates an overhead of 288kps and a latency >1.98s; efficient sampling and classification approach (ESCA) [3] generates an overhead of 215kps and a latency of 1.98s.

To keep pace with an ever-changing set of requirements, network device has rapidly become programmable across the board: from switches to network interface cards (NIC) to middle boxes. These data planes are referred as programmable data plane (PDP). PDPs, such as programmable switch (e.g., Intel Tofino) and smartNIC (e.g., Netronome Agilio CX 2x10GbE [4], Mellanox MCX654106A-ECA) provide very high performance with processing latencies that can reach nanoseconds. PDPs have some computing and memory resources to support deploying ML. Furthermore, PDPs are the closest place to packets. Thus, deploying ML on PDPs has some unique advantages, such as, quickly responding to network dynamics, and avoiding the traffic overhead caused by sending packets to the control plane or outside analyzers.

However, in order to achieve forwarding packets at line rate, PDPs hold limited operations and programming model (e.g., missing loop), and can only give little memory and computation resource to support application-specific tasks offloaded. PDPs do not support complex operations required by ML, such as multiplication, polynomial or logarithms. Therefore, how to deploy ML on PDPs is a challenge.

While programmable devices have been proven to be useful for in-network computing, there are few successful cases of implementing ML on PDPs. Considering the computing and memory requirements of ML, most researches about deploying ML on PDP are simple non-neural network(NN) ML. For example, flow classification [5] and anomaly detection [6] by decision tree, support vector machine, clustering and random forest. But these simple non-NN ML cannot achieve satisfactory performance, such as low classification accuracy.

Ideally, the performance of NN with more layers is better than that of non-NN ML. However, compared to non-NN, NN require larger memory, more operations and programming model. At the cost of sacrificing little precision, binary neural networks (BNN) [7] require low memory, and its computation can be performed using lighter mathematical operations, such as bit shift. For programmable switch and smart-NIC, programming protocol-independent packet processors (P4) is current the most widespread abstraction and programming language. Specially, P4 program has good portability and can be ported across hardware platforms. Thus, this article proposes P4-BNN (Binary Neural Network based on P4). Our contributions are follows:

(1). We propose a framework that uses P4 to completely execute binary neural network on PDPs. Where we validate our framework by flow classification and anomaly detection use-case which are deployed on a Netronome smartNIC. For comparison, we also implement N3IC, decision tree by if-else chain and K-means with Manhattan distance on this smartNIC.

(2). We propose a tailor-made data structure suitable for deploying BNN on PDP, such that the input layer can use bit shifts and simple integer arithmetic operations to replace multiplication. Where the input of decimal features is converted to be binary. All 1-bit weights in a neuron are spliced into a decimal value, which is stored by the registers to reduce the times of reading/writing registers.

(3). We propose a normalization method in PDP which needn't floating-point operations. Its key idea is that only caring about positive or negative of normalized results rather than their specific value.

(4). We propose a method of updating parameters of BNN model in runtime, where registers are used to store the weight parameters of the BNN model. Since the registers can be conveniently and quickly read and written from the control plane, the model can be updated in runtime.

II. Related Work

Non-NN on PDPs. IISY [8] introduced a network packet classification system on PDP. They explore packets classification using in-network supervised and unsupervised ML algorithms (decision trees, K-means, SVM, and Naïve Bayes) implemented in P4. [9] proposed to deploy a decision tree into the programmable data plane by matching action table for flow classification, but both of them use a lot of memory to store the table. [10] proposed an ML component in the control plane to convert the decision tree model to P4 language by if-else chain and deployed it into Netronome smartNIC to actually measure resource consumption. [11] proposed to deploy the random forest algorithm by matching action table in the programmable data plane to realize the task of attack detection. The stateful and stateless features with more complex calculation are used, but the performance results depend on the number of registers or other storage elements. DeepMatch [12] allow stateless intra-packet and stateful inter-packet (i.e., flow-based) deep packet inspection (DPI) on the NPs-based smartNIC, employing regular expression matching. Instead of only packet header inspection shown in other works, DeepMatch delivers line-rate regular expression matching on packet payloads.

NN on PDPs. N2Net [13] and BaNaNa [14] have made significant contributions to offloading deep learning (DL) processing in the programmable data plane. N2Net designed for embedded applications that run on resource constrained devices and thus require simple arithmetic operations. However, it is not deployed and tested in the real network. BaNaNa proposes to spilt the input layers of NN on a CPU. The remaining layers go through a quantization process that converts the original NN model into a format that can be run on programmable network devices. However, this method adds communication latency between CPU and PDP. N3IC [15] use Micro-C language to implement the neural networks on Netronome SmartNIC with simple bit operations, but it has not yet provided normalization layer for neural networks.

We completely converted the BNN into the P4 language, and deployed all neural network layers onto PDP, and provide a more detailed process of deploying BNN on PDP.

III. Background and Motivation

A. SmartNIC

The NIC communicates with the host's CPU and GPU through PCIe. This process involves four times PCIe transfers, which often takes a lot of time to transmit data in delay-sensitive network tasks. Some lightweight tasks can be offloaded to the smartNIC, which not only reduces the data processing latency, but also releases the CPU resources of the host.

B. Binary Neural Networks

(1). Equation (1) is the standard for the binarization of weight parameters. A weight value (w) occupies only one bit. Compared to the floating-point weight matrix, the memory consumption of the network model can be reduced 32 times than before.

$$\operatorname{sign}(w) = \begin{cases} +1, & w \geq 0 \\ -1, & \text{others} \end{cases} \tag{1}$$

(2). When the weight value and the activation function value are binarized at the same time, the multiplication and addition operation of 32 floating-point numbers can be solved by the simple operation of bitwise (XNOR), and population count (POPCOUNT).

We conducted validation experiments to evaluate the compressed model size of BNN and the computational complexity of the inference process. For MNIST dataset to use LeNet-5 model, before binarization, the model size is 173KB and the accuracy reaches 98.6% during the inference process, and the calculation amount is 282KMACs (MAC represents a Multiply-accumulate operations). After binarization, the model size is reduced to 7KB and the accuracy can also reaches 97.0%. More important, the multiplication operation in the inference process can be replaced by simple arithmetic operations of bit.

IV. P4-BNN System Design

In this section, we describe the challenges of deploying DL on PDP, and how do P4-BNN to address them.

A. Challenges of Neural Networks on PDP

Challenge 1. Small model is desired. A typical smartNIC has ~10 MB memory, which is mainly used to store packet information, and host forwarding and policy tables. Thus, the memory left for NNs is very small. However, NNs often have lots of parameters, which require a large of memory.

Challenge 2. Simple operations are desired. The computing capabilities of network devices is limited. PDPs often only perform bitwise logic, shift and simple integer operations. However, NNs require a lot of complex operations (e.g., matrix operations, multiplication, division, floating-point, and loop).

B. P4-BNN overview

The P4-BNN architecture is shown in Fig.1, which converts the NN inference into P4 and is deployed on a smartNIC to achieve flow classification and anomaly detection. Where all operations for flows are done on the data plane instead of the control plane. This can avoid latency caused by sending flows information to the controller.

Based on P4-BNN, we design two types of classification models. One is the packet-level model which employs features from individual packet, and another is flow-level which employs features from flows. Specially, one advantage of flow-level model is the ability of capturing the continue network dynamics, such as cumulative flow duration and flow length at a given moment, which cannot be accomplished only relying on single packets. They have richer features, so the flow-level model performance will be better than that of packet-level. However, the flow-level model requires retaining each flow information to store their respective feature values. The packet-level model need not store anything, so there are no overhead associated with flow information, such as memory consumption. However, packet-level model requires the ML to process each packet, and this will may lead to long processing latency. These infers will be confirmed by the latter experiment results.

Control Plane. It is responsible for collecting flow information from the data plane to build training dataset and training the BNN model. This dataset is processed into binary encoding (Section IV-C) for training. After the model training, the control plane send new model parameters to the smartNIC's registers to update the weight of the model.

Data Plane. It receives commands and parameters from the control plane, such as accepting new model parameters. When the packet enters the smartNIC, it extracts packet information from the packet header and then performs BNN inference, normal packet forwarding, and updates flow information.

When a packet enters the smartNIC, for flow-level model, the processing includes the following four steps. For packet-level model, only the third and fourth steps need to be performed, but BNN operations are done for each packet, as shown in Fig.1.

(1). **Hashing**. We use hash to mark each packet with the flow ID which it belongs to, and then look up the *result register* according to the flow ID. Where the inputs of hash are some header fields (i.e., IP.src, IP.dst, Port.src, Port.dst, IP.protocol). If the classification result exists in the *result register* (The classification results are obtained according to the first *N* packets of the flow), it will be written to the ToS field of IP header for forwarding. Otherwise, next step is the *feature gathering*.

(2). **Feature gathering**. It extracts the relevant features of the packet and stores them into *feature register*. *N* is a threshold value that indicates the number of packets needed to be collected. If the counter reaches the *N*, the *BNN executor* is triggered to perform the BNN inference. Otherwise, the packet will be normally forwarded without classification result.

(3). **BNN executor**. It executes BNN inference to classify flow. The classification process is as follows. First, the *feature register* is read to get flow features, and then these features are inputted to the BNN model for inference. Finally, the inference result is assigned to the flow and used for updating *result register*. The classification result is written into ToS field of IP header.

(4). **Forwarding**. According to the classification result, selecting different ports to forward the packets.

C. Design Details of P4-BNN

(1). Achieving P4-BNN Inference on PDP

The P4-BNN inference process is shown in Algorithm 1. The basic operations are XNOR, POPCOUNT, NORMALIZATION and SIGN. The computation between neurons use XNOR operation which is a simple shift. SIGN is a comparison operation. NORMALIZATION is an operation without floating-point, as shown in line 4 to line 13. For POPCOUNT, we count the number of 1 in bit string. Specially, at line 3, the $W_k^b \oplus a_k$ represents XNOR between the binary weight of bth neuron at kth layer and input of kth sample to obtain a bit string. Then performing POPCOUNT operation to obtain the number of 1 in the bit string (*i.e.*, *c*). Next, performing NORMALIZATION operation to make the linear activation value in the same distribution (*i.e.*, *d*). Finally performing SIGN operation to obtain the final result (*i.e.*, SIGN(*d*)) which is a value of 1 bit (0 or 1). Because there are multiple neurons, this operation is repeated for many times, in other words, there are multiple *d*. We splice these SIGN(*d*) into a bit string as the input of next layer, as shown in line 14 of Algorithm 1. More details are shown in *calculation between neurons* of Fig. 2.

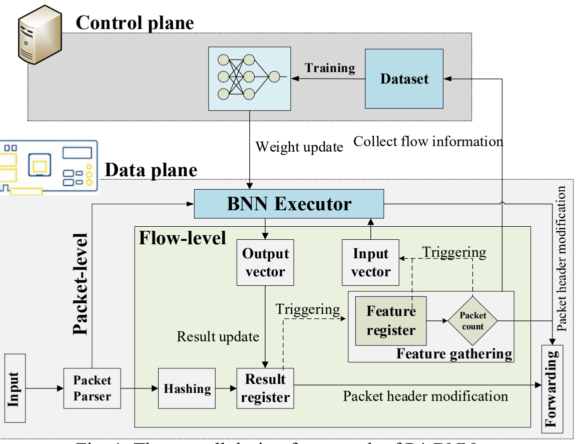


Fig. 1. The overall design framework of P4-BNN

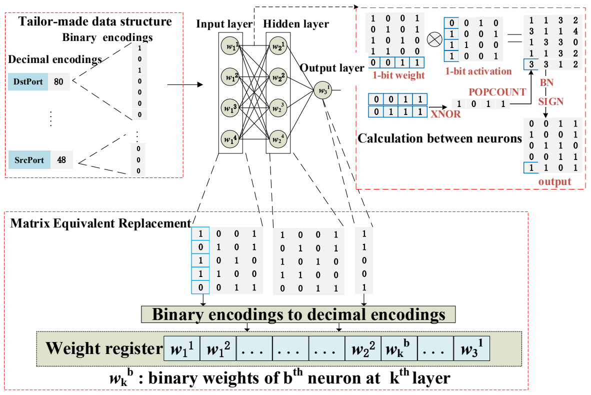
Algorithm 1 P4-BNN Inference.

```
Input:
  $a_k$ : binary input of kth layer
  $W_k^b$ : binary weights of bth neuron at kth layer
   $\lceil\mu_k\rceil$: mini-batch mean of kth layer
Output:
  $y$ : prediction results
1: for k = 1: K-1 do   //$K$ is the number of layers
2:  for b = 1 : B do   // $B$ is the number of neurons in each layer.
3:    c ← POPCOUNT($W_k^b \oplus a_k$) //$c$ is the number of 1 in the bit string, $\oplus$ is XNOR operation.
4:    if $c \geqslant t_k$   //$t_k$ is half the length of the bit string.
5:      if $sign$ ==0 // $sign$ is the sign bit of $\lceil\mu_k\rceil$
6:        $d = 2 * (c - t_k) - \lceil\mu_k\rceil$
7:      else
8:        $\mathrm{SIGN}(d) = 0$
9:    else
10:     if $sign$ ==0
11:       $\mathrm{SIGN}(d) = 0$
12:     else
13:       $d = |\lceil\mu_k\rceil| - 2 * (t_k - c)$
14:  $a_{k+1} \leftarrow (a_{k+1} \ll 1) + \mathrm{SIGN}(d)$
15: end for
16: y ← $\mathrm{SIGN}(\mathrm{POPCOUNT}(W_K^1 \oplus a_K))$
```

(2).Tailor-made Data Structure and Matrix Equivalent Replacement

The inference of BNN can use XNOR and POPCOUNT instead of multiplication, but this method is only suitable for hidden layers. For the input layer, the input is usually in decimal form. For such inputs, XNOR and POPCOUNT operations cannot be used to speed up the inference process. In other words, the non-match between input layer and hidden layers maybe slow down the speed of inference. Therefore, P4-BNNprocess input data by binary, i.e., converting the value of each feature from decimal to be binary. For example, the source ports and destination ports are a 16-bit integer in the PDP, as shown in *tailor-made data structure* of Fig. 2. This processing turns the one original input into multiple inputs. While this will increase the number of inputs, it can let the input layer to be accelerated by simple operations, which naturally match with the characteristics of PDP.

Each weight of P4-BNN model has only 1 bit with values 0 or 1 (P4-BNN uses 0 instead of -1). Thus, during inference, we use registers to store the weights of model. As shown in *Matrix Equivalent Replacement* of Fig. 2, to reduce the times of reading/writing registers, all 1-bit weights in a neuron are spliced into a decimal value stored by register. Each value of register represents the weight of a neuron. Besides, in normal programming, matrix operation can be realized by loop. However PDP does not support loop operation, we use multiple reading/writing registers to equivalently replace matrix operation between neurons. The times of reading /writing registers are same as the number of neurons.



Tailor-made data structure
Binary encodings
Decimal encodings
DstPort 80
SrcPort 48
Input layer
Hidden layer
Output layer
1-bit weight
1-bit activation
POPCOUNT
XNOR
BN
SIGN
Calculation between neurons
output
Matrix Equivalent Replacement
Binary encodings to decimal encodings
Weight register
w_k^b : binary weights of b^th neuron at k^th layer

Fig. 2. BNN Executor

(3). Batch Normalization in PDP

In terms of accelerating and stabilizing the training of neural networks, it has been proved that the normalization is an effective technology . However, the floating-point operations required in the normalization make it difficult to be implemented in PDP. Batch normalization (BN) [16] use trainable parameters γ and β to retain the nonlinear features learned from activation. We observe that BN in P4-BNN can eliminate floating-point operations by set γ=1 and β=0. In the hidden layers, subtracting the mini-batch mean (μ) from the linear activation value is the main operations of normalization, and then the result of subtraction will be divided by the root of the mini-batch variance ($\sqrt{\sigma^2+\varepsilon}$). Therefore we only need to care about the positive or negative of normalized results rather than their specific values which will be binarized (0 or 1). Because $\sqrt{\sigma^2+\varepsilon}$ is always positive, the division operation can be ignored. We use ⌈μ⌉ to approximate μ, where the sign is the highest bit.

In summary, BN includes three steps, (i) checking whether the *c* is greater than the threshold (*t*); (ii) checking the sign bit of the ⌈μ⌉; (iii) linear activation value subtract the ⌈μ⌉ as the input of SIGN. As shown in line 4-line 13 of Algorithm 1.

(4). Updating BNN Model in Runtime

Because the traffic pattern vary greatly, the classification performance maybe worse if a unchanged BNN model is used to classify these dynamic traffic. Therefore, to improve the generalization of P4-BNN model, how to update model in runtime is an important issue. In the P4-BNN, the control plane can conveniently and quickly read and write the registers, so P4-BNN can real-time collect new packet features by reading the features registers and uses some registers to store the weights of BNN model. Thus, after training a new BNN model, the control plane can update the BNN model by modifying the weight stored in the register (i.e., writing the registers), when the P4 program is running.

V. Use Case

Flow classification. UNI1 [17] is a widely used dataset to evaluate flow classification. The flow types can be classified from the dimensions of flow length, flow duration, flow speed, etc. This article uses the flow length as the classification basis. For UNI1, we set the threshold of flow length to 10KB. So, when the flow length is greater than 10KB, it is an elephant flow, otherwise it is a mouse flow. According to this criterion, we sorted out 277130 flows, including 63298 elephant flows and 213832 mouse flows. After splitting the dataset, 80% of the flows is used for training and 20% is used for testing. The proportions of flows in each class were set to be the same as in the whole dataset.

For flow classification, we also uses the flow length and flow duration as the classification basis to achieve four classification. If the time interval between two packets in the same flow is greater than 60s, the following packets will form a new flow. According to this criterion, we observe that most of the flow length are less than 2KB and most of the flow duration are less than 1s. Table I shows the threshold used when building four-classifications.

Anomaly detection. The CIC-IDS2017 [18] contains benign and the most up-to-date common attacks, which resembles the true real-world data (PCAPs). This includes network traffic from Monday to Friday. We selected the PCAP files from Wednesday and divide it into attack and benign network flow. we sorted out 60000 data flows, including 30000 attack flows and 30000 benign flows. After splitting the dataset, 80% of the flows is used for training and 20% is used for testing.

Features analysis. How to select the suitable features is one key issue for flow classification and anomaly detection . We determine features by considering the cost of PDP extracting feature and scoring base on random forest. Where the higher score it gets, the better the relation between the feature and flow type [19]. Table II shows the features used when building packet-level model and flow-level model in P4-BNN. For flow-level four classification, we added flow duration as feature.

TABLE I. Threshold of four types of flows in UNI1

	Flow length	Flow Duration	Number of flows
Class 0	⩽ 2KB	⩽ 1s	133411
Class 1	⩽ 2KB	> 1s	42884
Class 2	> 2KB	⩽ 1s	62987
Class 3	> 2KB	> 1s	37848

TABLE II. PACKET-LEVEL AND FLOW-LEVEL FEATURES

Model	Feature	Short description
Packet-Level	DstPort	TCP/UDP destination port
	SrcPort	TCP /UDP source port
	Packet length	Length of the packet
	TTL	Time to live
Flow-Level	DstPort	TCP/UDP destination port
	SrcPort	TCP/UDP source port
	Packet length	Length of the packet
	Flow length	Length of the flow since the arrival of the first packet

VI. EXPERIMENTS

A. Experiment setup

Testbed. We benchmark P4-BNN in a simple two node topology. The Netronome smartNIC (Agilio CX 2x10GbE) deployed P4-BNN is installed in a Sugon I420 server with dual Intel Xeon 420 10-core 2.60 GHz processors and 16 GB DDR3 1600MHz RAM. It is connected via 10 GbE cables to a traffic generation server. The traffic generation server is also a Sugon I420. The traffic generation server uses *tcpreplay* command to send these packets to the virtual interface vf0-1.

Inside the smartNIC, each packet was handled according to three versions of P4 application. (i) Baseline, no ML algorithm is deployed; (ii) ML model of packet-level; (iii) ML model of flow-level. Each model structure of P4-BNN is a regular Multilayer Perceptron (MLP) with 3 fully connected layers (FC) of 8, 16, 1, where also with 2 normalization layers. More detai ls are shown in Table III.

TABLE III. DETAILS OF P4-BNN MODEL STRUCTURE

Class	Input size (bits)	NN size (neurons)	Memory (KB)
Packet-Level	56	8,16,1	0.258
Flow-Level	128	8,16,1	0.338

Comparison. We realized an ablation about the true effect of batchnormalization in the scene (N3IC [15]), the decision tree (DT) with an if-else chain [10] and K-means measured by Manhattan distance on smartNIC are compared with P4-BNN. Of course, DT and K-means can also be realized with the match-action table, but it requires a lot of memory. Besides, full precision model based on artificial neural network (ANN) achieved on server CPU is regarded to a performance baseline, which model struct is same as P4-BNN.

B. Experiment results

Table IV contains accuracy, precision and recall on the flow classification and anomaly detection, where we observe the firs t 5 packets(i.e., N=5) in the flow. We observe that the ANN has the best performance, as expected. Our proposed P4-BNN method has only slightly lower accuracy than ANN. Because binarization loses the information carried by the weights, but the P4-BNN can compress the memory required for weights by 32 times compared with the ANN. P4-BNN performs better than N3IC, because the batch normalization layer can improve the accuracy of the neural network. We can also observe that whether packet-level or flow-level, P4-BNN shows higher performance than DT and K-means model. Take flow classification as an example, we can see that the accuracy of the flow-level is higher than the packet- level. Because the flow-level contains the information of multiple packets, it has richer features.

Furthermore, we also evaluate P4-BNN for four-classification task. Table V shows that the accuracy of ANN, P4-BNN, N3IC and K-means all greatly decrease. The behind reason is follows, the four-classification task requires more feature/weight information than two-classification task, however the weight of P4-BNN lose too much information after binarization. When P4-BNN use a same model structure, fitting the data for four-classification task is obviously more difficult than two-classification task. To address this challenge from four-classification task, designing a more complexly model structure for P4-BNN maybe a possible method, and we will do it in the future.

TABLE V. RESULTS OF FLOW-LEVEL FOR FOUR CLASSIFICATION.

Method	Accuracy	Precision	Recall	F1
ANN	77.3%	72.3%	70.4%	0.713
P4-BNN	55.4%	44.1%	46.9%	0.455
N3IC	48.8%	37.1%	26.2%	0.307
K-means	48.2%	34.7%	25.2%	0.292

C. Overhead

Packet processing latency. In order to evaluate the packet processing latency added by deploying ML, we measure the time difference between packet enters the physical port of smartNIC and the packet reaches virtual interface vf0-1. Fig. 3 shows the distributions of the time that a packet resides inside the smartNIC according to flow-level and packet-level for flow classification. We can see that the packet processing latency has increased after ML algorithm are deployed on the smartNIC.

The processing latency of the P4- BNN packet-level model is the largest. The main reason is that although P4-BNN packet-level classification does not require updating flow information and storing any data, but it will require operations for each packet (XNOR, POPCOUNT, SIGN and BN). The overhead of P4-BNN flow-level model is mainly caused by updating flow information. After collecting enough features for P4-BNN model, the remaining packets do not need to execute the P4-BNN inference process, but only need to find the result register to get the classification result. It can be seen that the overhead of P4-BNN inference process is higher than that of updating flow information.

For flow-level, the overhead of DT, K-means and P4-BNN models are almost the same, because their overhead is mainly caused by updating traffic information. For packet-level, we observed that the overhead of P4-BNN is much higher than DT and K-means, because P4-BNN requires more complex operations rather than a simple if-else chain. K-means needs to calculate the Manhattan distance, so the processing delay will be larger than DT, but the calculation of the distance is not as complex as the inference process of P4-BNN, so the processing delay will be shorter than P4-BNN.

Flow classification scheme based on P4-BNN can help flow scheduling on switch. Compared with making decisions in the control plane [2, 3], the increased processing latency of P4-BNN can be ignored.

Memory of code. For the Netronome smartNIC Agilio CX 2x10GbE, the entire application must be described in 8K instructions (or 16K in shared-code mode). Table VI shows the amount rate of code instructions for flow classification. Where Local Memory (LM) and cluster local scratch (CLS) represent

TABLE IV. SUMMARY OF RESULTS

	Flow classification								Anomaly detection							
	Flow-Level				Packet-Level				Flow-Level				Packet-Level			
	Accuracy	Precision	Recall	F1	Accuracy	Precision	Recall	F1	Accuracy	Precision	Recall	F1	Accuracy	Precision	Recall	F1
ANN	92.9%	95.3%	95.6%	0.954	87.8%	75.7%	65.7%	0.734	96.6%	99.6%	93.4%	0.964	87.8%	92.1%	93.3%	0.927
P4-BNN	91.7%	93.6%	94.7%	0.942	85.8%	66.8%	57.6%	0.619	96.4%	99.5%	93.2%	0.962	86.8%	90.9%	93.5%	0.922
N3IC[15]	90.0%	88.6%	89.0%	0.888	84.6%	70.9%	45.7%	0.556	95.7%	99.6%	91.7%	0.955	85.5%	90.5%	92.2%	0.914
DT [10]	90.6%	94.1%	93.8%	0.939	81.6%	70.9%	35.2%	0.471	93.7%	99.9%	87.4%	0.932	83.4%	83.5%	99.8%	0.909
K-means	71.4%	96.8%	65.1%	0.777	65.5%	40.0%	95.8%	0.563	48.7%	47.5%	51.6%	0.495	49.5%	84.0%	48.7%	0.616

the memory space of the code in the smartNIC. We observe that DT and K-means has a low code instructions due to its simple implementation. P4-BNN needs more code instructions due to its simple implementation. P4-BNN needs more code instructions, due to it is more complex than DT and K-means. It needs to do some calculations like XNOR, POPCOUNT, SIGN and BN between neurons.

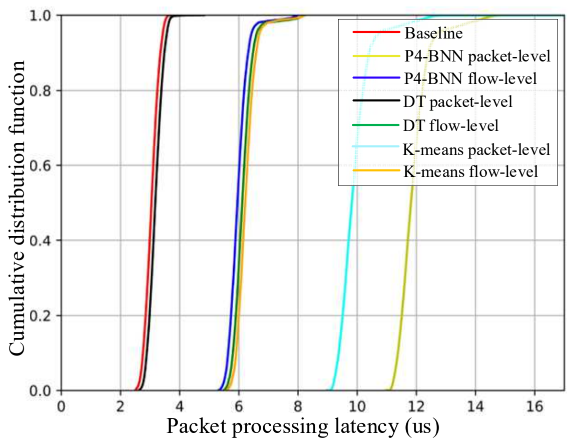


Fig. 3. Cumulative distribution function of packet processing latency at the smartNIC.

TABLE VI. THE AMOUNT RATE OF CODE INSTRUCTIONS

	Baseline	P4-BNN Packet-level	P4-BNN Flow-level	DT Packet-level	DT Flow-level	K-means Packet-level	K-means Flow-level
CLS	23%	28%	39%	23%	28%	27%	34%
LM	39%	49%	69%	39%	49%	49%	59%

VII. CONCLUSIONS AND FUTURE WORK

In this article, we propose a method using BNN for in-network classification, solving the problem of deploying neural networks in the programmable data plane, and deploy it into Netronome smartNIC to validate our ideas. The results show that in flow classification and anomaly detection, the accuracy reaches more than 91.7% and 96.4% respectively.

Furthermore, since single PDP is difficult to support full ML, a distributed deploy framework for ML maybe a promising method where distributing the neurons of an neural network (NN) into multiple switches. In this direction, in-network neural networks [20] has presented some interested use-cases, such as, network telemetry and anomaly/intrusion detection.

ACKNOWLEDGMENT

This work was supported by the National Natural Science Foundation of China (61872102, 61972104, 62272113) the Guangdong Basic and Applied Basic Research Foundation (2021A1515012306, 2018A0303130045), Guangzhou Key Laboratory of Software-Defined Low Latency Network (202102100006), China.

REFERENCES

[1] R. Boutaba, M. A. Salahuddin, N. Limam, et al, “A comprehensive survey on machine learning for networking: evolution, applications and research opportunities,” Journal of Internet Services and Applications, vol. 9, no. 1, p. 16, 2018.

[2] Chao S C, Lin K C J, Chen M S. "Flow classification for software-defined data centers using stream mining" [J]. IEEE Transactions on Services Computing, 2016, 12(1): 105-116.

[3] Tang F, Zhang H, Yang L T, et al. "Elephant flow detection and differentiated scheduling with efficient sampling and classification" [J]. IEEE Transactions on Cloud Computing, 2019:1-15.

[4] NETRONOME. (2019) Netronome Agilio SmartNIC. [Online]. Available: https://www.netronome.com/products/agilio-cx/

[5] Eric Liang, Hang Zhu, et al. Neural Packet Classification. In Special Interest Group on Data Communication (SIGCOMM). ACM, 2019.

[6] Georgios Kathareios, Andreea Anghel, Akos Mate, et al. Catch It If You Can:Real-Time Network Anomaly Detection with Low False Alarm Rates. In International Conference on Machine Learning and Applications (ICMLA). IEEE,2017.

[7] Qin H, Gong R, Liu X, et al. Binary neural networks: A survey[J]. Pattern Recognition, 2020, 105: 107281.

[8] Zheng C, Xiong Z, Bui T T, et al. IIsy: Practical In-Network Classification[J]. arXiv preprint arXiv:2205.08243, 2022.

[9] Kamath R, Sivalingam K M. Machine Learning based Flow Classification in DCNs using P4 Switches[C]//2021 International Conference on Computer Communications and Networks (ICCCN). IEEE, 2021: 1-10.

[10] Xavier B M, Guimarães R S, Comarela G, et al. Programmable switches for in-networking classification[C]//IEEE INFOCOM 2021-IEEE Conference on Computer Communications. IEEE, 2021: 1-10.

[11] Lee J H, Singh K. SwitchTree: in-network computing and traffic analyses with Random Forests[J]. Neural Computing and Applications, 2020: 1-12.

[12] Hypolite, Joel, et al. "DeepMatch: practical deep packet inspection in the data plane using network processors." Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies. 2020.

[13] Siracusano G, Bifulco R. In-network neural networks[J]. arXiv preprint arXiv:1801.05731, 2018.

[14] Sanvito D, Siracusano G, Bifulco R. Can the network be the AI accelerator?[C]//Proceedings of the 2018 Morning Workshop on In-Network Computing. 2018: 20-25.

[15] G. Siracusano, S. Galea, D. Sanvito, et al. Rearchitecting Traffic An-alysis with Neural Network Interface Cards, USENIX Symposium on N etwork-ed Systems Design and Implementation, NSDI 2022.

[16] Ioffe S, Szegedy C. Batch normalization: Accelerating deep network training by reducing internal covariate shift[C]//International conference on machine learning. PMLR, 2015: 448-456.

[17] Theophilus Benson, Aditya Akella, David A. Maltz. Network Traffic Characteristics of Data Centers in the Wild[C]// Acm Sigcomm Conference on Internet Measurement. ACM, 2010.

[18] Sharafaldin I, Lashkari A H, Ghorbani A A. Toward generating a new intrusion detection dataset and intrusion traffic characterization[J]. ICISSp, 2018, 1: 108-116.

[19] W. Liu, J. Cai, Y. Wang, et al, “Fine-grained flow classification using deep learning for software defined data center networks,” Journal of Network and Computer Applications, 2020,168(10).

[20] Luizelli M C, Canofre R, Lorenzon A F, et al. In-Network Neural Networks: Challenges and Opportunities for Innovation[J]. IEEE Network, 2021, 35(6): 68-74.