An Accurate & Efficient Approach for Traffic Classification Inside Programmable Data Plane

Muhammad Saqib*, Zakaria Ait Hmitti*, Halima Elbiaze*, Roch H. Glitho†

* Université du Québec à Montréal, Montreal, Canada
† Concordia University, Montreal, Canada
{saqib.muhammad,ait_hmitti.zakaria}@courrier.uqam.ca, elbiaze.halima@uqam.ca, glitho@ece.concordia.ca

GLOBECOM 2022 - 2022 IEEE Global Communications Conference | 978-1-6654-3540-6/22/$31.00 ©2022 IEEE | DOI: 10.1109/GLOBECOM48099.2022.10000863

***Abstract*—In-network traffic classification is a class of in-network computing that brings significant benefits to the network, i.e., the first line of defence, classification at line rate and fast reaction time. However, it is still challenging to accurately and efficiently classify Internet traffic at an early stage due to a clear trade-off between flow identification time and classification accuracy - both are competing objectives. To this end, we introduce a framework that focuses on deploying an accurate network traffic classifier inside a programmable data plane that can classify the traffic at maximal speed while considering the underlying constraints of the device. Notably, we move from statistical feature-based traffic analysis and argue that traffic flow can be classified using a single feature called sequential packet size information as input. We evaluate our approach by identifying different types of IoT traffic inside a programmable data plane. Our findings demonstrate that accurate and early-stage network traffic classification is achievable with minor use of networking device resources.**

***Index Terms*—machine learning, programmable data plane, P4, in-network computing, in-network classification**

I. Introduction

Generally, in-network classification is a class of In-Network Computing (INC) [1] and inspired by the reconfigurability of the match-action paradigm [2]. With the rise of INC [3], [4], the interest is rapidly growing to run Machine Learning (ML) algorithms inside Programmable Data Plane (PDP) [5]–[7]. Running ML models inside the networking device significantly impacts the network. First, switches offer very high performance. The latency through a switch is in the order of hundreds of nanoseconds per packet [5]. Second, the performance of distributed ML is bounded by the time required to get data to and from nodes. So, if a switch can classify the traffic at the same rate that it carries packets to nodes in a distributed system, then it will equal or outperform any single node.

Last but not least, the networking device can serve as the first line of defence by terminating unnecessary data close to the edge. As a result, it can help save energy, reduce traffic load on networking infrastructure, and improve user experience by lowering communication latency. Latency-sensitive applications will significantly benefit. Packet and flow are the two core objects to extract from headers and payloads in the network traffic classification process. The work in [6] investigated the trade-off between a per-packet or a per-flow based classification model. Per-packet model proved to be more efficient but less accurate, while the per-flow one is contrariwise. Since accuracy and efficiency are two competing objectives, the value of both cannot be overstated. The information needs to be aggregated from several packets in the per-flow case. Having these richer features lead to better training of the model. However, keeping up-to-date flow states in the memory is a resources intensive solution that needs extra memory and demand for complex operations to derive useful information from the aggregated data. A per-packet model is potentially more efficient that can classify the traffic at a line rate without updating any features in the memory. However, it does not offer aggregated measurements and the possibility of learning from temporal correlation to the model. As a result, due to a clear trade-off between detection time and classification accuracy, it is still challenging to perform accurate and early-stage network traffic classification.

In addition, there are certain limitations at PDP, such as the lack of support for complex operations and a limited amount of memory (tens of megabytes) to store many features for the incoming flows. Hence, setting up an upper bound on the performance of the traffic classifier inside PDP. Therefore, the design of a classification model that fits the constraints of PDP (e.g., no floating points, no loops, and limited memory) is challenging. Consequently, it is necessary to use the minimum number of features to reduce lookup and update overhead and identify the flow earliest stage while respecting the underlying constraints at the data plane.

This work aims to propose an accurate and efficient in-network traffic classification approach subject to the data plane constraints. Instead of using several statistical features of the flow, which are memory intensive and need complex operations, we only take a single feature as an input. Our evaluation results show that the proposed solution can identify the traffic type for several source applications at an early stage of the flow creation.

The remaining of this work is organized as follows. Section II presents the background information and our work position with the literature. Section III represents the proposed solution. The system validation is shown in Section IV. Finally, Section V represents the concluding remarks and future directions.

This work was fully supported by CHIST-ERA program under the "Smart Distribution of Computing in Dynamic Networks (SDCDN)" 2018 call.

II. Related Work

This section aligns our proposed solution with state of the art. In recent years, there has been a rising interest in research combining ML and networking. For instance, recent works such as [8] investigated the problem of ML-based traffic classification. However, few considered the data plane programmability facet. For instance, papers [5], [6] use statistical properties analysis of the flow for traffic classification but do not consider the limitations at PDP. Therefore, we discuss the commonly used flow classification techniques and highlight their deficiencies. Finally, we highlight the importance of an accurate and efficient network traffic classification approach in next-generation programmable networks.

A. Flow classification techniques

Statistical-based flow analysis is a widely used technique for distinguishing network traffic by identifying differences in statistical properties of the flows [8]. Several packets must be tracked to obtain more detailed information about the flow. Sampling approaches were used to select a few packets for each flow and send them to the control plane, which hosts the classifier [9]. The choice of sampling rate, on the other hand, is crucial because it is highly dependent on the application requirements. A low sampling rate may result in a high rate of miss-classification, whereas a high sampling rate may overwhelm the controller with additional traffic overhead. In addition, the main flaw of this approach, which does not incorporate the learning process, is the static construction of such model [8]. In a dynamic networking environment, this flaw has a significant impact on the model's performance. Moreover, feature such as inter-arrival time is under time-domain measurement. Instability is the main problem with a time-domain measurement that it is always prone to performance degradation in dynamic network conditions. As a result, statistical-based traffic classification approaches are limited in their ability to handle the dynamics in next-generation high-precision networks due to these robustness flaws.

B. P4-switch as a classification machine

Naturally, the switch acts as a classification machine. Upon receiving an object (a packet), the switch first extracts the relevant features from its headers, such as IP, port, protocol type, packet size, etc. The parser extracts these fields where each field is itself a feature. The switch keeps these extracted features inside a Packet Header Vector (PHV) and then applies the pipeline process to the vector. Based on such motivation, the work in [5] demonstrates the mapping of trained ML algorithms to reconfigure able match-action tables (RMTs) [10]. Generally, the training module generates the resulting outputs in a decision tree where the control plane API called P4Runtime embeds the outputs into the switch's RMTs. The authors validate their work by classifying IoT traffic based on some statistical properties of the flows; however, they did not examine accuracy or efficiency. Another work investigates a clear trade-off between traffic identification time and classification accuracy inside PDP [6]. Since packet and flow are the two core objects of classification decisions, it is hard to decide when picking one another. The per-flow based classification process is costly in terms of classification latency and resource consumption. Also, the demand for complex operations limits its applicability to PDP. On the other hand, there is no need to keep track of the packets in the per-packet case. Therefore, the resources overhead are not present anymore. However, it is hard to accurately identify the traffic class based on very little independent feature information provided by individual packets. Consequently, bringing accuracy efficiency to the network traffic classification process is still challenging and highly desirable in next-generation high-precision networks.

III. System Design

This section presents the necessary steps to deploy an ML model into a programmable networking device by using P4 language. Fig. 1 shows the high-level in-network classification architecture. The control plane characterises the source traffic and maps the resulted output into the data plane for online inference[1].

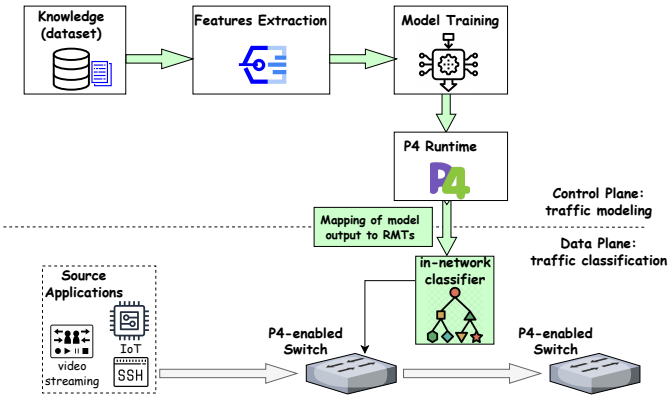

Fig. 1: In-network traffic classification architecture

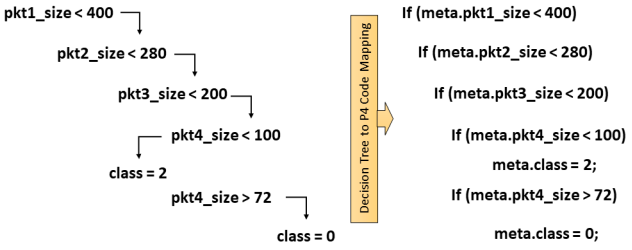

Fig. 2: Mapping of decision tree to P4 code

A. Offline data training

In this phase, the control counterpart trains ML models on a given dataset and translates them into target switches for traffic identification at runtime. This section focuses on ML model training while considering the requirements such as flow identification time, classification accuracy and limitations of P4.

Flow and metrics: A flow f_i is a sequence of packets p_j having the same five tuples (IP addresses, ports, protocol

[1]Our source code is publicly available at https://github.com/em-saqib/inc

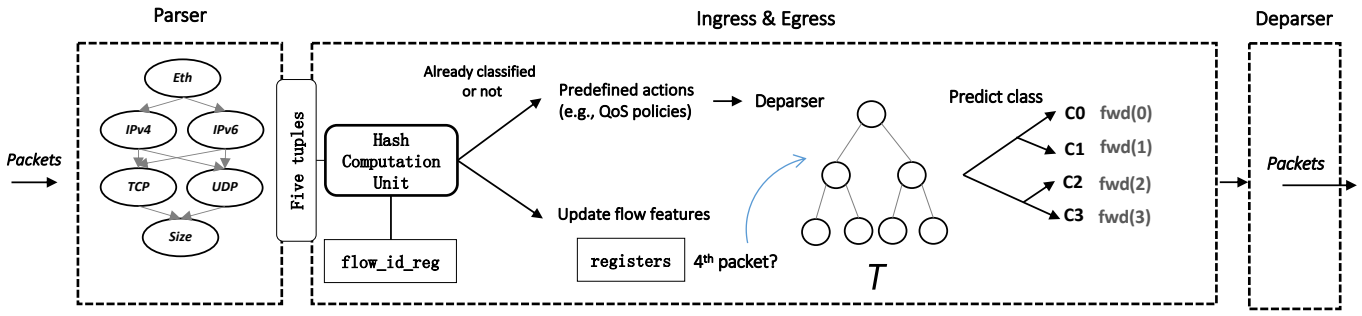

Fig. 3: Packet processing pipeline in the data plane

type). The first j packets of the flow f_i are denoted $f_i(1:j)$. The source dataset contains various traffic flows having different Quality of Service (QoS) requirements. Flows with similar QoS requirements belong to the same class, and the flow identification process for different QoS groups is known as multi-class classification. Two metrics are used to assess the flow classification accuracy: true positive rate (TPR) and true negative rate (TNR) [11].

Traffic features: TABLE II lists the features used to train ML models. The packet size is used for a single feature and the rest for multi-features model.

1) Feature extraction

The two core objects in network traffic classification are $flow_i$ and p_j, which are used to extract information from traffic headers and payloads. The flows are mainly identified based on the statistical properties of traffic. However, keeping up-to-date flows' states in the switch is a resources intensive solution that needs extra memory and demand for complex operations to derive useful information from the aggregated data. In addition, the features derived from time-related metrics such as inter-arrival time may not be consistent and stable enough to serve the classifier in dynamic network conditions [12]. Also, network conditions such as bursty heavy loads and traffic congestion may affect the time-related metrics. To solve this problem, we used the most stable feature, namely packet size information, as input for the classifier [13].

2) ML model training

The original dataset S_i consists of packets of subflows $(f_i(1:j))$ and is further split into training S_i^T and testing S_i^P samples. After preparing the dataset, the next step is to characterize the traffic by applying the ML algorithm. There are many supervised learning approaches in the literature, but not all are appropriate for our work. In other words, since we aim to embed the ML model's output into the data plane, the necessary operations in the targeted model must be readily available in P4. Therefore, we decided to use a decision tree algorithm to cope with the P4 limitations. Given the P4 language's current primitives, a decision tree classifier is more suitable for such a task. Only comparison operation is required to classify an element x, and it can be easily expressed in P4 using *if-else* statements (see Fig. 2). T_i is defined as a decision tree that predicts the flow class after receiving the j^{th} packet for a flow f_i.

3) ML to P4 converter

A decision tree algorithm makes decisions based on the values of input parameters (i.e., features) and can be represented with a tree structure. When using a decision tree to classify an element x, one must traverse the tree from root to leaf, respecting the conditions in each node until a leaf node is reached. This procedure can be easily implemented in general-purpose computer languages using recursion or repeating loops. In the P4 language, however, neither of these alternatives is available. As a result, hard-coding the tests and labels within the tree-nodes into *if* and *else* statements is an option. To that end, the ML to P4 component translates the model's *if-else* conditions into a P4 code that describes the generic behaviour for a given application.

B. Online inference

A general process of online inference is shown in Fig. 3. The switch maintains a few registers to record $flow\_id$, *packet_sizes*, *packet_counter* and other statistics such *min/max/avg packet size* and *total number of bytes* for the first few packets of each flow. The diagram depicts how each packet traversing the device is handled. For the incoming packets, the parser module extracts the relevant features (*i.e., five tuples and packet size*) from the header and keeps these features' values in the pipeline's metadata. The next step is to calculate the hash value for each flow based on the *five tuples* field from the packet's header. The *flow_id* register keeps track of all classified flows to treat the belonging packets accordingly. For the incoming packets belonging to identified classes, the switch applies corresponding actions. In other words, the classified flows' packets will not necessarily go through the decision tree process and will be processed at a line rate. In the event where the flow is not classified, the switch verifies *packet_counter* for the corresponding flow. Until the *packet_counter* reaches the *threshold*, the parser extracts *packet_size* and stores it into a *size_vector*. Once the *packet_counter* meets the *threshold*, a classification occurs on the *size_vector* in following the *if-else* chain, encoded in P4 code. The flow's class will eventually be saved in the *meta.class* variable. The detailed steps are revealed by

Algorithm 1: Online inference

```
Input: TCP and UDP packets, thr: max # of packets
Output: classes_vector
classes_vector = []; actions_vector = []; flow_id=[];
 size_vector=[];
min_size= ∞ ; max_size=0; avg_size=0; total_bytes=0;
Function InferClass(packets):
    while packets do
        flow_id = hash(five_tuple);
        if isClassified(flow_id) then
            ApplyAction(flow_id);
        else
            if pkt_counter <thr then
                single_feature(flow_id, pkt_size, pkt_
                counter, size_vector) ;   // Algorithm 2
                multi_features(flow_id, pkt_size, min_size,
                max_size, avg_size, total_bytes) ;
                 // Algorithm 3
                pkt_counter ++
            if pkt_counter == thr then
                classes_vector[flow_id] =
                Apply_SF_Model(flow_id, size_vector);
                 classes_vector[flow_id] =
                Apply_MF_Model(flow_id, features_vector);
End Function
Function isClassified(flow_id):
    if classes_vector[flow_id]! = 0 then
        return True;
End Function
Function ApplyAction(flow_id):
    egress_port = actions_vector[flow_id];
End Function
```

algorithms (1-3).

Algorithm 2: Single feature

```
Input: flow_id, pkt_size, packet_counter, size_vector
Output: size_vector[flow_id]
size_vector[flow_id * 4 + packet_counter] = pkt_size;
Return size_vector[];
```

Algorithm 3: Multi features

```
Input: flow_id, pkt_size, min_size, max_size, avg_size,
       total_bytes
Output: min_size, max_size, avg_size, total_bytes
if pkt_size<min_size then
    min_size = pkt_size;
if pkt_size > max_size then
    max_size = pkt_size;
total_bytes = total_bytes + pkt_size;
if packet_counter == thr then
    avg_size = Extern_Division(total_bytes, thr);
Return min_size, max_size, avg_size, total_bytes;
Function Extern_Division(total_bytes, thr):
    division_result = total_bytes/thr;
Return division_result;
End Function
```

IV. Performance Evaluation

This section evaluates our proposed in-network traffic classification solution by considering a use case of IoT devices generating data traffic belonging to various QoS groups. We demonstrate the efficiency of our solution explicitly in accurately identifying the class of source devices at an earliest stage inside PDP with minimal usage of device resources.

We use packet capture (PCAP) traces of IoT devices released by [14] as our dataset. From the available dataset instances, we selected the PCAP files for nine days (from 22 to 30 Sep 2016), containing flows related to five applications comprising different IoT devices. Since we aim to identify the source devices based on a single feature (i.e., packet size), we only select the packet size feature for our ML model training which can be directly extracted from the header.

TABLE I: Dataset summary

Device	Class	# of flow	# of packets
Amazon Echo (AE)	Smart assistants	16788	270840
Security Camera (SC)	Cameras	1601	144187
Motion Detector (MS)	Smart home devices	6411	233329
Photo-Frame (PF)	Appliances	5439	23561
Weather Station (WS)	Sensors	979	11623

TABLE II: Selected features

Feature	Type	Short Description
SrcPort	Stateless	Source port
DstPort	Stateless	Destination port
Pkt_size	Stateful	Size of the packet
Min_pkt_size	Stateful	Size of the smallest packet
Max_pkt_size	Stateful	Size of the largest packet
Avg_pkt_size	Stateful	Average packet size of flow
Total_bytes	Stateful	Cumulative sum of IPv4 packet size

1) Dataset

We divide the monitored devices to five classes: static smart-home devices (e.g., motion detector), sensors (e.g., weather station), audio (e.g., smart assistants), video (e.g., security camera), and appliance (e.g., photo frame). We select classes that can be assigned to various QoS groups: from high bandwidth (video) and low latency (sensor) to best effort (others classes). The devices belonging to the same class sharing the same traffic characteristics. Therefore, we select only one device from each class for validation in the data plane. TABLE I shows a summary of the dataset for these selected devices.

2) Experimental setup

The experimental procedure starts from training the ML model on the given IoT dataset to embedding the trained model' outputs to the data plane for online inference. We also observe the CPU and memory overhead added by the ML model to the P4 switch during online inference.

In order to compare the performance of our single feature-based traffic classification approach with a statistical method, we train two different ML models based on the features described in TABLE II. The statistical properties allow the capture of flow dynamics (e.g., duration and cumulative packet size at a given moment). However, the downside is extra resources overhead and limited support of crucial operations (e.g., division and square root) in P4. Therefore, it is impossible to compute better descriptive measures directly (e.g., average, variance, and standard deviation) for time-varying features when working with flows. In a single feature case, we

can directly extract the packet size from the header and the only operation required to infer class is a value comparison.

The first step is to train ML models on selected samples from the dataset. We are using Python's scikit-learn [2] implementation of the decision tree classifier to build the models. The training set for both single and multi-feature models is the same. However, the feature extraction process differs in both cases. In addition, the input length (i.e., number of packets) and decision tree depth must be kept to a minimum to identify the flow class at the earliest possible stage. Therefore, we are observing the input length and depth of the tree to understand the impact on classification accuracy. The obtained results show that with a tree depth of four and an input length of four, the single-feature model provides good accuracy (99%) as shown in TABLE III. As a result, both parameters are set to four.

The next step of the experiment is to test the models in the data plane for online inference. We are applying both single and multi-feature models to all the packets in the test set. The data plane is implemented in P4, compiled with a target of behavioural model version 2 (BMv2) [15]. Moreover, another experiment step is to assess the CPU and memory overhead added by both models to the regular packet processing to accomplish the actions required for classification (feature extraction, updating feature values, and identifying the flow class).

TABLE III: Varying input length and tree depth

	Input Length				
Tree Depth	**1**	**2**	**3**	**4**	**5**
1	55%	55%	72%	72%	75%
2	60%	60%	76%	84%	84%
3	71%	71%	83%	91%	91%
4	73%	73%	73%	**99%**	99%
5	73%	75%	90%	99%	99%

TABLE IV: Classification results

	Single-feature model			**Multi-features model**		
Class	Precision	Recall	F1-Score	Precision	Recall	F1-Score
AE	0.99	1.00	0.99	1.00	1.00	1.00
MS	1.00	1.00	1.00	1.00	1.00	1.00
PF	0.99	0.99	0.99	1.00	0.96	0.98
SC	1.00	0.92	0.96	0.94	0.91	0.99
WS	0.98	1.00	0.99	0.98	1.00	0.99

3) Results

This subsection presents the obtained results, including the classification accuracy, the class identification time and the resources overhead added by ML models to the data plane.

Classification accuracy: TABLE IV summarizes the classification results of both single and multi-feature models using the following performance metrics: precision, recall and f1-score. It is clearly shown that both models performed similarly for class identification. The confusion matrix for online inference of both models is shown Fig. 4, with the accuracy of both models being similar.

[2]https://scikit-learn.org/

Class identification time: This performance criterion concerns packet residence time inside the switch's pipeline. It can be obtained by calculating the packet forwarding latency throughout the pipeline processing. Equation 1 is used to calculate the identification time T_{f_i} of a particular flow f_i. It is the sum of j^{th} packet processing time and the time spent in feature extraction and values updating process by unclassified packets $(1 : j-1) \in f_i$.

$$T_{f_i} = T_t^{j \in f_i} + T_t^{(1:j-1) \in f_i} \tag{1}$$

The average flow identification time for each class is shown in Fig. 5. In the multi-features case, the class identification time increases with the number of features. This can be explained by the necessary operations for calculating statistical properties and updating corresponding flow entries in the memory.

Classification cost: the last set of results concerns the cost of network traffic classification inside PDP in terms of CPU and memory consumption of the switch. Again, it is slightly higher in the multi-features case, indicating that an increase in the number of features directly impacts the device's memory and computational resources. (See TABLE V).

TABLE V: Classification cost

Model Type	**CPU**	**Memory**
Single feature	6.1%	0.39%
Multi-features	7.4%	0.49%

4) Discussion

The obtained results reveal that instead of classifying the flow based on statistical properties that are prone to network dynamics, affect the class identification time and adds-up extra resources overhead at the device; we can accurately and efficiently identify the network traffic only based on the sequential packet size information of the first few packets by each flow. Since we are using packet size information as an input, the applications having large-sized messages such as high-quality video streaming will cause fragmentation at the network layer, affecting the overall class identification time. However, as shown in TABLE VI, the payload size in emerging and latency-critical IoT applications is less than the maximum transmission unit (MTU). As a result, our proposed approach is potentially applicable to high precision networks where the latency-critical applications will benefit greatly. Meanwhile, taking a single feature as the input increases the system utilization in terms of CPU and memory consumption. In addition, despite having good accuracy, the one fit model becomes outdated due to the changing traffic pattern. At the same time, the network device memory needs to be carefully maintained. Therefore, it is vital to continuously monitor the network device, remove inactive match-action rules from the device memory, and use the telemetry data to train the model better to keep an updated model in the data plane. All these limitations and considerations are subject to our future work.

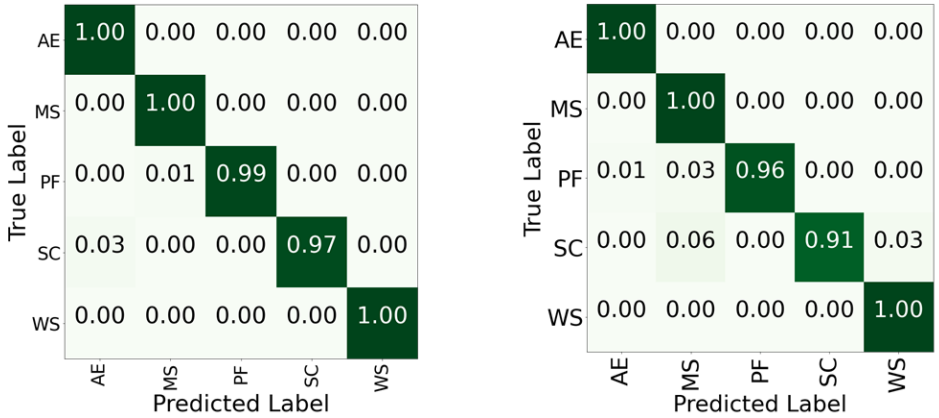

Fig. 4: Confusion matrices for single feature (left) and multi-feature (right) model

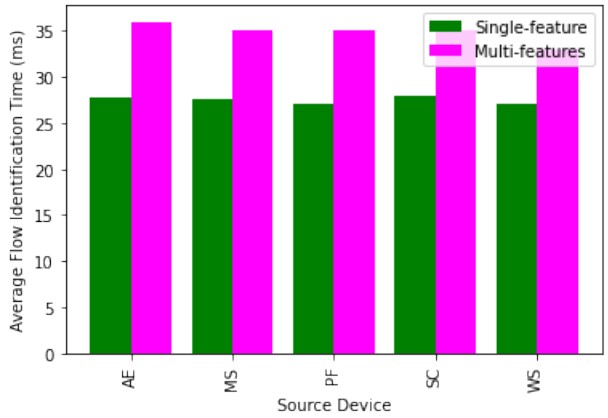

Fig. 5: Class identification time

TABLE VI: Payload size and latency requirements of IoT applications in next-generation networks [16], [17]

Use Case	E2E Latency (ms)	Data Size (bytes)
Factory Automation	0.5 to 50	10 to 30
Process Automation	50 to 100	40 to 100
Smart Grids / 2.0	3 to 20 / 1 to 10	80 to 1000
Intelligent Transportation	10 to 100	<500
Internet of Everything	ms to s	-

V. Conclusions

In this paper, we presented an accurate and efficient approach for network traffic classification inside PDP that can identify various types of IoT traffic at an early stage by considering only a single feature as an input. Our proposed solution mainly consists of two phases, (i) offline model training at the control plane and (ii) online inference in the data plane. The first phase uses a data training algorithm to identify the traffic source based on a single feature, while the second phase uses the resulted outputs as match-action rules inside PDP to identify the flow class at runtime. The simulation results show that accurate and early-stage network traffic classification is achievable inside PDP by considering only the first few packets' sizes information of each flow as an input. Our proposed solution achieves high precision early-stage network traffic classification, allowing efficient differentiated QoS provisioning.

References

[1] Y. e. a. Tokusashi, "The case for in-network computing on demand," in *EuroSys Conference*, 2019, pp. 1–16.
[2] P. e. a. Bosshart, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 99–110, 2013.
[3] I. Kunze, K. Wehrle, D. Trossen, and M.-J. Montpetit, "Use cases for in-network computing," *IETF, Internet-Draft*, 2021.
[4] Y. Tokusashi, H. Matsutani, and N. Zilberman, "Lake: the power of in-network computing," in *IEEE International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, 2018, pp. 1–8.
[5] Z. Xiong and N. Zilberman, "Do switches dream of machine learning? toward in-network classification," in *ACM workshop on hot topics in networks*, 2019, pp. 25–33.
[6] B. M. e. a. Xavier, "Programmable switches for in-networking classification," in *IEEE Conference on Computer Communications*, 2021, pp. 1–10.
[7] X. e. a. Zhang, "pheavy: Predicting heavy flows in the programmable data plane," *IEEE Transactions on Network and Service Management*, vol. 18, no. 4, pp. 4353–4364, 2021.
[8] F. e. a. Pacheco, "Towards the deployment of machine learning solutions in network traffic classification: A systematic survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1988–2014, 2018.
[9] S. Sadrhaghighi, M. Dolati, M. Ghaderi, and A. Khonsari, "Flowshark: Sampling for high flow visibility in sdns."
[10] P. e. a. Bosshart, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.
[11] P. Poupart, Z. Chen, P. Jaini, F. Fung, H. Susanto, Y. Geng, L. Chen, K. Chen, and H. Jin, "Online flow size prediction for improved network routing," in *2016 IEEE 24th International Conference on Network Protocols (ICNP)*. IEEE, 2016, pp. 1–6.
[12] J. Erman, A. Mahanti, M. Arlitt, I. Cohen, and C. Williamson, "Offline/realtime traffic classification using semi-supervised learning," *Performance Evaluation*, vol. 64, no. 9-12, pp. 1194–1213, 2007.
[13] W. e. a. Chen, "Sequential message characterization for early classification of encrypted internet traffic," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 4, pp. 3746–3760, 2021.
[14] A. Sivanathan, H. H. Gharakheili, F. Loi, A. Radford, C. Wijenayake, A. Vishwanath, and V. Sivaraman, "Classifying iot devices in smart environments using network traffic characteristics," *IEEE Transactions on Mobile Computing*, vol. 18, no. 8, pp. 1745–1759, 2018.
[15] "Performance of bmv2," https://github.com/p4lang/behavioral-model/blob/main/docs/performance.mdwhat-impacts-performance, online; accessed 213 February 2022.
[16] C. e. a. De Alwis, "Survey on 6g frontiers: Trends, applications, requirements, technologies and future research," *IEEE Open Journal of the Communications Society*, vol. 2, pp. 836–886, 2021.
[17] P. e. a. Schulz, "Latency critical iot applications in 5g: Perspective on the design of radio interface and network architecture," *IEEE Communications Magazine*, vol. 55, no. 2, pp. 70–78, 2017.