K-Flix: Clustering-Based Video Streaming Traffic Identification in Programmable Data Planes

Amaury Teixeira Cassola[1], **Alberto Schaeffer-Filho**[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

{atcassola, alberto}@inf.ufrgs.br

Abstract. *Given the demand for video streaming on the Internet, specific Quality of Service (QoS) solutions become necessary, the implementation of which depends on the timely identification of such traffic. With this goal, this work uses Clustering for video traffic classification by offloading a Nearest Centroid Classifier to the data plane. A prototype of the proposed system was developed in a virtual environment and tests were executed using real traffic captures. Accuracies of over 98% were achieved on all tests, demonstrating the potential of the technique.*

1. Introduction

Video streaming applications accounted for over 65% of global Internet traffic in 2022 [Sandvine 2023]. Given such a demand, specific QoS policies to manage video streaming traffic would be especially useful. However, the execution of application-specific QoS solutions depends on the timely identification of the traffic in the network. The research community has proposed multiple solutions for Internet traffic classification over the years, with recent ones relying on the analysis of statistical attributes of packet flows. In particular, numerous research efforts have been developed using Machine Learning (ML) technologies to map between behavior patterns and application classes [Nguyen and Armitage 2008]. Although promising results have been observed, severe limitations have also been noted, especially regarding system efficiency. Generally, such systems show a significant communication overhead, since all packets need to be duplicated and forwarded to a server where the classification occurs.

Recent advances in Programmable Data Planes (PDPs) have opened up new possibilities for the application of ML in traffic classification, with several recent works exploring this method and obtaining encouraging results with little or no impact on network performance and high classification efficiency [Parizotto et al. 2023]. In this context, there is relatively little research focusing on the Unsupervised Learning paradigm, especially if compared to models such as Decision Trees and Neural Networks [Parizotto et al. 2023]. Furthermore, the recent literature insufficiently investigates the specific classification of video streaming, even though it is among the most relevant traffic classes.

This paper proposes K-Flix, an in-network ML system capable of classifying packet flows as either video streaming or not. To this end, the P4 language [Bosshart et al. 2014] is used to map a classifier defined by the centroids of a previously trained K-Means model to the pipeline of a programmable switch. The switch, in turn, calculates the statistical attributes of the packet flows and classifies them at runtime. The system was evaluated using traffic captures of Video on Demand (VoD) applications and achieved over 98% accuracy in identifying the downstream video flows with a low amount of false positives.

The main contributions of this paper can be summarized as follows:

- The design and implementation of a cluster-based classification system for video streaming traffic in programmable data planes.
- The proposal of a novel in-network normalization approximation algorithm for ML models in programmable switches.
- The prototype and evaluation of K-Flix in a virtual emulated scenario.

The remainder of this paper is organized as follows. Section 2 provides a brief overview on both programmable data planes and the use of clustering for traffic classification. Section 3 describes the design of K-Flix. Section 4 details the prototype implementation and Section 5 discusses the results of the system evaluation. Lastly, Section 6 presents the related work, and Section 7 concludes the paper.

2. Background

In this section, we describe the background on programmable data planes and discuss the use of clustering techniques for traffic classification.

2.1. Programmable Data Planes

Traditional network devices are implemented using Application-specific Integrated Circuits (ASICs). The ASICs employed in such devices offer the necessary throughput, but are generally of fixed function, presenting little to no programmability [Hauser et al. 2023]. More recently, hardware abstraction models were developed with the aim of exposing the packet-processing functionalities to high-level programming languages. One example of such models is the Protocol-Independent Switching Architecture (PISA), which abstracts packet processing as a pipeline of Reconfigurable Match Tables (RMTs) [Bosshart et al. 2013].

The PISA model is divided in three programmable stages: a parser which deserializes ingress packets and extracts data from their headers; a processing pipeline which uses match-action tables to process and modify packet data and, lastly, a deparser responsible for serializing a packet and sending it to an output port. Based on this model, the P4 language is currently the most used programming language for describing data plane algorithms [Hauser et al. 2023]. It allows programming each of the stages of the architecture and the population of the RMTs. Given the need for line rate processing, P4 does not support dynamic memory allocation, recursion, loops and floating point operations such as division and logarithms.

2.2. Clustering for Traffic Classification

Network traffic classification consists of identifying the applications responsible for packet flows through the analysis of traffic features. The use of traffic classification techniques is essential for a number of management and security operations such as intrusion detection, performance monitoring, and the generation of statistical reports on the network usage [Boutaba et al. 2018].

Unsupervised learning consists of identifying patterns and structures in unlabeled data sets. In the context of traffic classification, it is rare to have information

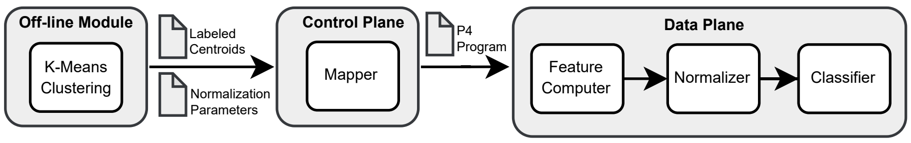


Figure 1. K-Flix System Overview

regarding all applications present in a network, so clustering techniques can be used to find patterns in packet flows and subsequently identify the corresponding applications [Boutaba et al. 2018]. Particularly, classifiers based on the output of the K-Means clustering algorithm have demonstrated promising results in this context [Usama et al. 2019]. The centroids generated by the model can be combined with the K-Nearest Neighbors algorithm, resulting in the Nearest Centroid Classifier, which classifies instances according to the label of the closest centroid, simplifying the inference process. Being a relatively simple algorithm with low computational cost, it is especially appropriate for use in PDPs

3. K-Flix Design

K-Flix is an in-network traffic identifier capable of classifying unidirectional packet flows as either video streaming or not. It uses centroids generated by a K-Means model to implement a Nearest Centroid Classifier [Zhang and Sun 2010] which is mapped to the processing pipeline of a programmable switch.

3.1. Approach Overview

Our goal is to achieve traffic classification of video streaming applications based on unsupervised learning output in programmable data planes. To this end, there are a number of challenges that must be overcome. Given that the P4 language does not support complex constructs such as loops, the direct implementation of a machine learning model is difficult, so it is necessary to define a mapping process from the classifier to an appropriate program and structures. Furthermore, normalization of the computed features is necessary, but there is no support for divisions in P4, which are fundamental for most normalization algorithms. Therefore, there is a need to develop a new approach for feature normalization that is not based on divisions.

Figure 1 presents an overview of K-Flix. The system implements a Nearest Centroid Classifier which assigns labels to instances based on the centroids generated by a clustering algorithm. Given the need to minimize resource usage, the clustering of the data and the generation of the classifier happen offline, with only the inference stage being offloaded to the data plane to be executed in real-time. The system consists of three subsystems. The *offline module* is responsible for using a K-Means model to cluster traffic captures and generate the centroids that will be used by the classifier. In addition, it performs the analysis of the features to generate the parameters necessary for normalization. The *control plane* maps the previously generated data to a P4 program and a sequence of match-action tables. The classifier is deployed to the *data plane*, which is responsible for performing the inference process. At runtime, the flows traversing the programmable switch are analyzed to extract relevant statistical attributes and classification is performed at line-rate. Next, we present details about our design.

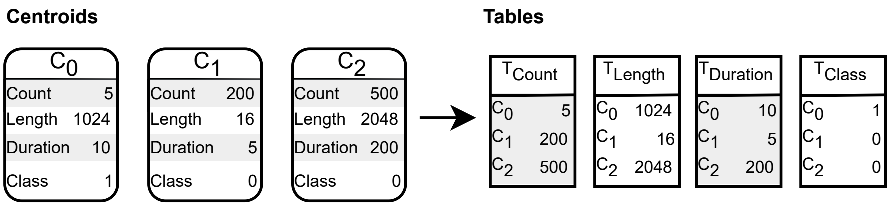


Figure 2. Mapping of the K-Means Classifier to Match-action tables

3.2. Nearest Centroid Classifier in Programmable Data Planes

In a K-Means Nearest Centroid Classifier, the inference algorithm consists of calculating the similarity between the instance to be classified and each of the centroids of the partitions, and choosing the cluster with the centroid closest to the input. Using the Euclidean Distance as a similarity measure, the calculation of the distance between an input x and a centroid C^i is defined by Equation 1. Considering k centroids, the inference consists of calculating each of the distances $\{D_1, D_2, ..., D_k\}$ and choosing the cluster whose centroid results in the smallest distance.

$$D_i = \sqrt{(x_1 - C^i_1)^2 + (x_2 - C^i_2)^2 + ..(x_n - C^i_n)^2} \tag{1}$$

Given a set of n features, K-Flix implements a classifier mapping based on the generation of intermediate distance values over $n + 1$ stages. At each stage of processing, the distances between a given feature in the input and each value of the same feature in the centroids are calculated, generating a one-dimensional distance vector. In the next stage, the distances for the next feature are calculated, and so on, accumulating the values in the one-dimensional distance vector. It is important to note that, when dealing with positive values and distances greater than zero, the Euclidean Distance square root operation can be ignored. In the last stage, the classification consists of choosing the cluster linked to the smallest calculated distance.

Similarly to [Xiong and Zilberman 2019], the algorithm is implemented in the form of n intermediate match-action tables, one for each feature, and a final table that performs the classification. Each intermediate table consists of a single entry that refers to the distance calculation function and includes the values of the corresponding feature in each of the centroids. At runtime, when one of these tables is applied, the distance between the corresponding feature in the analyzed flow and in each centroid is calculated. Along the pipeline, the distances are accumulated in a register in memory. The last table receives as input the labels corresponding to each centroid, compares all k distances between them, finds the smallest one and labels the flow with the corresponding class.

A mapping using this technique is illustrated in Figure 2. In this example, only three attributes are considered: number of packets, average packet length, and flow duration. In addition, three centroids are defined, with only the first one labeled with the class of interest. Thus, four tables in total are generated: one corresponding to each feature and a final table that performs the classification. It is important to note that the matching aspect of the tables is irrelevant in this application since each time a table is applied, the corresponding action must be invoked. Therefore, the matching keys and action names are omitted from the illustrations of match-action tables. In the execution of the inference in this example, a vector of intermediate distances kept in memory is updated as each table is applied, with the distances of each centroid being added to the total. The entire process is performed by sequences of simple arithmetic operations, in addition to comparisons between integers in the final step, which configures a very simple inference process appropriate for the context of PDPs.

3.3. A Normalization Approximation Algorithm

In machine learning, normalization consists of mapping all features to the same scale so that they all have the same weight in the final decision. One of the simplest normalization techniques is Min-Max normalization, which maps attributes to a scale defined from the maximum and minimum values of each one. The algorithm consists of applying Equation 2 to each value of each attribute X. Normalization must be applied both to the dataset, before training the model, and to each new instance to be classified.

$$x_{i\_norm} = \frac{x_i - X_{min}}{X_{max} - X_{min}} \tag{2}$$

However, given that Min-Max normalization relies on division, its implementation in PDPs becomes complex due to the lack of support for this operation. As a way to solve this problem, a novel algorithm for approximating normalization is proposed in this paper. Although division is not available, it is possible to perform bit shifts, which are equivalent to multiplications or divisions by powers of 2. The proposed algorithm is based on the idea of approximating the Min-Max normalization division through a series of divisions by powers of 2. It consists of two stages: the *generation of parameters* for normalization from the original values of the attributes and the *normalization* itself through consecutive bit shifts. The first stage of the algorithm, which calculates the powers of 2 that are necessary to approximate the normalization, can be executed offline as a bootstrap of an ML system in PDPs, and only the normalization of the values must occur in the programmable device.

Parameter Generation. The first stage of the algorithm receives as input the maximum and minimum values of an attribute, a value representing the upper limit of the desired normalized range and a threshold value defining the maximum number of powers of 2 that should be considered. Initially, the divisor of Equation 2 is calculated, which is then divided by the maximum limit of the scale. The resulting value is used to calculate the reciprocal of the divisor, *i.e.,* the result of 1 divided by it. The algorithm uses the fact that division by a number is equivalent to multiplication by its reciprocal. In the case of the reciprocal including an integer part in its value, this part represents a multiplication and will be returned as a multiplication factor. The reciprocal of the divisor is then iteratively approximated through reciprocals of powers of 2, with all powers appropriate for the approximation stored in a list. At the end, the list of powers and the multiplication factor are returned. The operation of this stage is presented in pseudocode in Algorithm 1.

In order to simplify the comprehension of the parameter generation stage, an illustration of the computation is presented in Table 1. Let's assume an attribute is considered whose Min-Max normalization on the traditional scale consists of dividing the attribute value by 25, so that the reciprocal of the divisor will be 0.04. In each row of the table, a

Algorithm 1: Normalization Parameters Generation

```
Input: FeatMax; FeatMin; ScaleMax, MaxPower
Output: Powers, list of powers of 2; MultFactor, multiplication factor
begin
    $Divisor \leftarrow FeatMax - FeatMin$;
    $ScaledDiv \leftarrow Divisor/ScaleMax$;
    $DivReciprocal \leftarrow 1/ScaledDiv$;
    $Multfactor \leftarrow$ integer part of $DivReciprocal$;
    $Decimal \leftarrow$ decimal part of $DivReciprocal$;
    $Powers \leftarrow \emptyset$;
    $CurPower \leftarrow 1$;
    $CurValue \leftarrow 0$;
    while $CurPower < MaxPower$ do
        $Temp \leftarrow CurValue + 1/(2^{CurPower})$;
        if $Temp \leq Decimal$ then
            Append $CurPower$ to $Powers$;
            $CurValue \leftarrow Temp$;
        end
        $CurPower++$;
    end
    return Powers, MultFactor
end
```

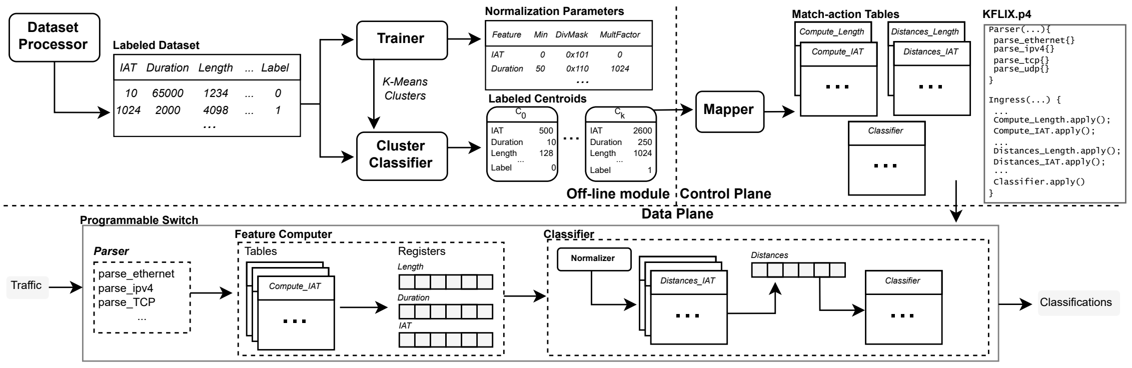
new power of 2 is tested. The reciprocal of this power is calculated and accumulated in a temporary variable. If the resulting value is less than the desired reciprocal, in this case 0.04, the corresponding power is added to the list and the temporary variable is updated.

Table 1. Values during parameter generation

Power	*Current Sum*	*List of Powers*
	...	
5	0,03125	$\emptyset$
6	0,046875	$\{5\}$
7	0,0390625	$\{5\}$
8	0,04296875	$\{5,7\}$
9	0,041015625	$\{5,7\}$
	...	

Normalization. Using the output from the previous stage, it is possible to calculate the approximate normalization of an attribute value in a programmable switch. Initially, the dividend of Equation 2 is calculated. For each of the powers in the list, a bit shift of a number of bits equal to the power is applied to the dividend. The results of each bit shift operation are accumulated, taking advantage of the fact that multiplication by a number X is equivalent to a series of multiplications by other numbers whose sum results in X. Finally, the value of the multiplication of the dividend by the multiplication factor calculated previously is also accumulated.

To illustrate this, a comparison between the approximate normalization performed with the proposed algorithm and the expected results obtained with Min-Max normalization on different attributes is presented in Table 2. A threshold of 16 powers of 2 and a desired range of values between 0 and 1,024 is considered. The use of this range in the context of PDPs is intentional: using an upper limit equal to a sufficiently large power of



```
Parser(...){
 parse_ethernet{}
 parse_ipv4{}
 parse_tcp{}
 parse_udp{}
}

Ingress(...) {
 ...
 Compute_Length.apply();
 Compute_IAT.apply();
 ...
 Distances_Length.apply();
 Distances_IAT.apply();
 ...
 Classifier.apply()
}
```

Figure 3. K-Flix Architecture

2 is equivalent to bit shifts to the left, which prevents the most significant bits from being lost in the division process. In the table, the values resulting from the approximation are compared with those returned by the original equation multiplied by 1,024. The proposed algorithm returns an appropriate approximation of the normalized values, presenting a very low average error.

Table 2. Approximate Normalization in different features

Min	*Max*	*Value*	*Min-Max Norm.*	*Approximation*	*Error*
5	96	75	787.6923	787	0.6923
0	1000	500	512	510	2
0	300	400	1365.3333	1363	2.3333
0	2000	200	102.4	101	1.4
0	1000	0	0	0	0

3.4. Integrated Techniques

In this section, we present additional design principles which were integrated into the K-Flix system.

Calculating Averages. Based on a technique proposed in our previous work [Coelho and Schaeffer-Filho 2022], an algorithm for calculating average values using consecutive bit shifts is implemented in K-Flix. The algorithm consists of calculating an approximate sum of the values considering a number of values equal to a power of 2, making it possible to calculate the average values of features.

Concurrent Flows Classification. By using a hashing algorithm, it is possible to map the attributes that represent a flow to a register position in memory. Thus, all packets belonging to the same flow are able to read and write from the same memory positions, which makes it possible to maintain and accumulate values related to multiple concurrent flows, a feature which is not natively supported by P4 targets.

3.5. Architecture

The K-Flix system is divided into a series of modular components. Figure 3 presents the organization of such components. In this section, the architecture of the system is discussed in detail.

Offline Module. Initially, the Dataset Processor analyzes traffic captures, identifies the flows present in them and extracts the attributes related to each one, labeling each flow as either video streaming or not according to a previous parameterizable classification. The Trainer, using the labeled dataset, performs two main functions: training a K-Means model and calculating the parameters required for the normalization approximation algorithm. Finally, in order to use the K-Means clusters to define the Nearest Centroid Classifier, it is necessary to associate a label to each of them. The implementation chosen for this purpose was the voting method, in which the label associated with a given cluster is the one that forms the majority among all the instances grouped in it. Thus, the Cluster Classifier analyzes the previously clustered instances and labels the centroids appropriately.

Control Plane. The only component present in the control plane is the Mapper. It maps the information generated by the previous components into structures appropriate for deployment in the data plane. Using the labeled centroids and the normalization parameters, this component algorithmically builds two distinct structures: the P4 program code and the commands for the population of the match-action tables. From the list of considered attributes, the attribute calculation tables are generated. Finally, the centroid attributes and the normalization parameters are used to create the traffic classification tables.

Data Plane. The Parser is implemented to support IPv4 packets. The Feature Computer calculates the features of each flow through a series of match-action tables, each responsible for a specific feature. The values are accumulated in memory registers indexed by hashes of the tuple of identifiers of each flow. In the case of a hash collision, indicating that a new flow will take the place of a previous one in memory, the accumulated data is reset and the current flow is simply considered a new flow. Finally, the Classifier component implements the Nearest Centroid Classifier, as described in Section 3.2. The flow classification is executed continuously with the classification being updated with every ingress packet. Before the classification itself, the Normalizer implements the second stage of the normalization approximation algorithm. After the classification, a K-Flix-specific header is added to the packet, comprised of two fields: *IsVideo*, with a binary value indicating whether the stream refers to a video application, and *Cluster*, with the cluster number corresponding to this classification, mainly for debugging purposes. The extra header approach was chosen both for simplicity of implementation and for the opportunity of continuous classification of the streams throughout their execution.

4. Implementation

A proof-of-concept prototype of K-Flix was developed for evaluation. The offline and control plane components were implemented using the Python language, with the K-Means model of the Scikit-learn library used to perform the clustering, while the data plane components were developed using the P4 language. The implementation is publicly available in the K-Flix GitHub repository [Cassola 2025].

The current prototype implementation analyzes a set of 9 traffic features listed in Table 3. The set of features was selected based on the analysis of previous works discussing relevant attributes for traffic classification with focus on video streaming identification [Zhang et al. 2013] [Dong et al. 2017]. Further analysis and optimization of the feature set is intended as future work.

Table 3. Selected Features

Feature	*Derivates*
Packet count	-
Packet Length	Sum, Max, Min and Mean
Inter-arrival times	Max, Min and Mean
Flow duration	-

Table 4. Train Dataset

App	*Flows*
Youtube	2,777
Netflix	634
Other apps	3,266
VoD flows	3,411
Total	6,677

For the Normalization Approximation algorithm, a scale of 2,048 possible integer values was chosen so that the normalized values could be represented as 11 bit strings. For the limit of the number of powers of 2 considered, we chose 32. This level of precision is necessary given the large scales of the selected attributes, especially the ones related to timestamps.

The feature extraction is implemented through 4 tables that store the values in 9 registers. The registers are hash tables with 32-bit entries for typical feature values and 48-bit entries for the timestamp-based features. To avoid hash collisions, the feature registers have a total of 65,536 entries. The classifier is implemented with one table for each feature and one extra table for the classification. The distances between the flow being classified and the centroids are kept in a register with 11-bit entries. In total, with 5 clusters, the current version utilizes 14 match-action tables and approximately 2.8MB of memory through 14 different registers. The optimization of the memory usage is intended as future work.

5. Evaluation

5.1. Experimental Setup

We conducted an experimental evaluation in a Linux VM with 6GB of allocated RAM running on an AMD Ryzen 5 processor with six cores. In this environment, the P4 program was compiled for a BMv2[1] virtual switch. The switch was configured to consume packets and forward them to virtual network interfaces, with the traffic being generated using *Tcpreplay*[2], a software tool capable of reproducing traffic from capture files. To capture the forwarded packets with the classification results, WireShark was used with a custom dissector to recognize the K-Flix-specific header. In this prototype, a unilateral flow is considered as a video streaming flow if it is a downstream flow from a video streaming application and can be considered an elephant flow [Dias Knob et al. 2017]. For the labeling of the centroids generated by the model, a voting technique was implemented in which the label of a cluster is the one that comprises the majority of that cluster's instances.

A publicly available traffic captures repository was used for the train and test datasets: *YouTube, Netflix, Web dataset for Encrypted Traffic Classification* [Shamsimukhametov et al. 2021]. It consists of traffic captures labeled by the application being accessed and includes usage of popular VoD streaming services YouTube and Netflix and a number of other miscellaneous applications. The training dataset contained 6,677 flows, of which 3,411 were identified as video streaming. The distribution of the

[1]http://bmv2.org
[2]https://tcpreplay.appneta.com

flows on the considered applications is presented in Table 4. For testing, 1,233 flows were sent to the virtual switch, of which only 54 could be considered video streaming flows according to the conditions adopted by this work. The total number of packets sent through the switch was 69,940, totaling approximately 61MB of data. The distributions of the flows in the test dataset can be seen in Table 5.

Table 5. Test Dataset

App	*Total Flows*	*VoD*
Youtube	314	31
Netflix	441	23
Other apps	478	0
Total	1,233	54

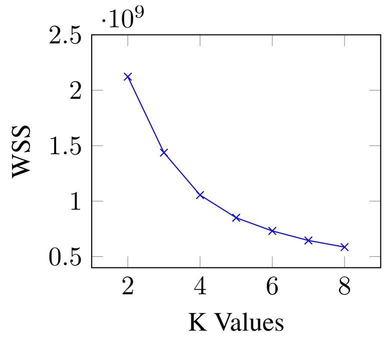


Figure 4. Elbow Method

5.2. Benchmarks

In order to evaluate and optimize the operation of K-Flix, a number of relevant metrics were considered. Accuracy, the ratio of correctly classified flows, with byte accuracy measuring the ratio of correctly classified bytes. Precision is the ratio of correct positive classifications, while Recall shows how many of the actual positives were correctly identified. Finally, the F1-score is calculated based on both precision and recall.

Given the necessity of determining the optimal value for the hyperparameter K, the Elbow Method was used on the Within-Cluster Sum of Squared Errors (WSS) of the generated clusters. Figure 4 presents the resulting plot of the WSS calculation for a varying numbers of clusters between 2 and 8. Given these results, the choice of the number of clusters to best minimize errors while avoiding overfitting would be around 5. To verify the correlation between cluster quality and the quality of the final classifications, three models with varying K values were trained and evaluated. Figure 5 presents a comparison of the classification results for each of the three models over the same workload.

While there is a clear increase in most metrics for $K = 7$, the decrease of the recall result shows a higher occurrence of false negatives. This effect can be correlated to the overfitting of the classifier as the number of clusters is increased, which damages the model's capacity of generalizing to unseen data. Furthermore, the amount by which the metrics improve between models diminishes in a manner very similar to the WSS values plotted in Figure 4, indicating that the quality of the classifier correlates directly to the quality of the generated clusters. From those results, it is possible to conclude that the optimal value of K for this model is 5. Figure 6 presents a more in-depth look at the results for the $K = 5$ model. Considering all 1,233 streams of the test workload, the accuracy obtained by the $K = 5$ model was 0.98, with a precision of 0.71, a recall of 0.83 and an F1 score of 0.77. In general, the results obtained were quite satisfactory and show promise for this approach.

The performance for both the considered video applications was very good, with most video streaming flows being classified correctly. That being said, there is a low but significant amount of false positives, that is, flows that were erroneously classified as

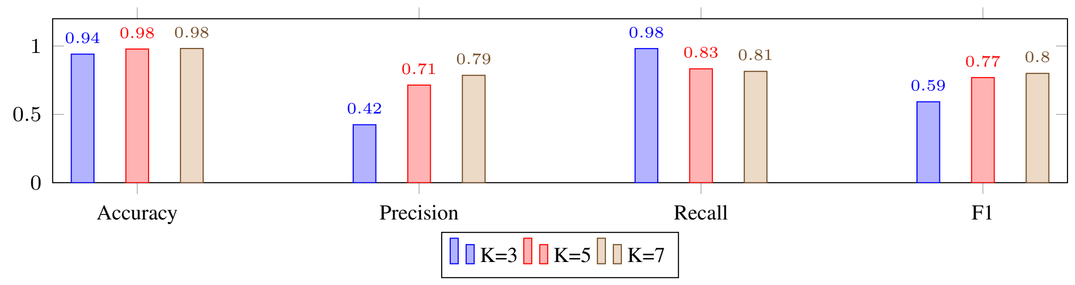


Figure 5. Overall classification metrics varying K

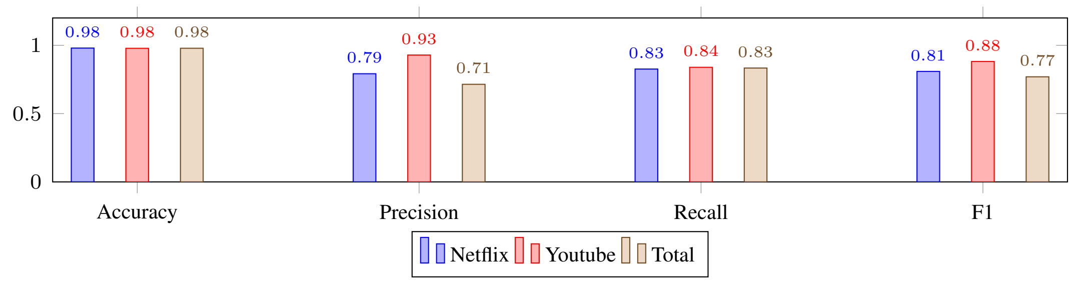


Figure 6. Classification metrics for K = 5

video streaming. Out of a total of 63 flows classified as video streaming, 18 were false positives. While these results are satisfactory, further optimization of the system is necessary before deployment in production environments. On the other hand, analyzing the negative classifications indicates a much better performance, with only 9 false negatives out of a total of 1,170 negative classifications. Given the nature of the video streaming use case, the metric of byte accuracy is especially relevant. It is defined as the accuracy of the classifier based on the volume of bytes that were correctly classified rather than the number of flows. The overall byte accuracy of the $K = 5$ model was of 0.84. When considering only the video streaming applications, the byte accuracy reached 0.98. This demonstrates that the model is able to classify flows correctly most of the time, but can struggle in differentiating video streaming for other types of downstream data transfer.

6. Related Work

Numerous recent works propose the offloading of ML inference to PDPs for either traffic classification or similar applications [Parizotto et al. 2023, Dalmazo et al. 2021, Nunes et al. 2023]. Table 6 presents a list of such related works.

In the context of machine learning in the data plane, Decision Trees (DTs) tend to be popular given the relatively simple inference process. [Xavier et al. 2022] propose a framework for the mapping of DTs to PDPs by translating them into chained *if-else* statements for traffic classification. The system presents promising classification results and low latency, but requires recompilations for model updates. On the other hand, [Coelho and Schaeffer-Filho 2022] map a Random Forests model into a series of reconfigurable match-action tables for the identification of Denial of Service attacks. The proposed system shows good results but does not account for the classification of multiple concurrent flows.

A few recent works employ unsupervised learning models for traffic clas-

Table 6. Works employing ML in PDPs

Work	*ML Technique*	*Application*	*Notes*
[Xavier et al. 2022]	Decision Trees	Traffic Classification	No runtime model updates
[Coelho and Schaeffer-Filho 2022]	Random Forests	DDoS Detection	No concurrent flows support
[Friedman et al. 2021]	Novel clustering system	Agnostic	Packet-by-packet classification
[Zhu and Zhang 2022]	K-Means and Decision Trees	Traffic Classification	Semi-supervised model
[Xiong and Zilberman 2019]	K-Means, among others	Agnostic	Packet-by-packet classification
[Cannarozzo et al. 2024]	K-Means	Agnostic	Train and Inference in the Data Plane

sification. [Friedman et al. 2021] present a classification system based on clustering using QuadTrees for the in-network inference process. Although the presented results are promising, the system is only able to classify individual packets. [Zhu and Zhang 2022] proposed a semi-supervised model utilizing both K-Means and Decision Trees and employing hashtables and Count-min Sketch techniques to store flow features. [Xiong and Zilberman 2019] investigate the mapping of various ML models to the data plane with the aim of optimizing resource usage. The authors conclude that the most viable mapping of a K-Means classifier would be creating one table per feature and an extra table for the classification. Lastly, [Cannarozzo et al. 2024] propose a system for the deployment of unsupervised models to SmartNICs, including both training and inference.

K-Flix differs from the discussed works in a few key aspects. Firstly, K-Flix aims to classify multiple concurrent flows by extracting and accumulating flow features, differing from both [Friedman et al. 2021] and [Xiong and Zilberman 2019] whose work calculates features on a packet-by-packet basis. Also, differently from [Zhu and Zhang 2022] who employ a semi-supervised model, K-Flix proposes a classifier based solely on an unsupervised learning model. Furthermore, [Cannarozzo et al. 2024] deploy both the training and inference stages to SmartNICs, while our approach intends to minimize resource usage by deploying only the inference process to the data plane. Lastly, K-Flix includes the definition of a novel algorithm for approximate feature normalization in the data plane, an aspect of machine learning which is not explicitly explored in the aforementioned works.

7. Concluding Remarks

This paper proposes K-Flix, a video streaming traffic classification system for programmable data planes. The system implements a Nearest Centroid Classifier based on K-Means clusters. Furthermore, a novel algorithm was proposed to approximate the normalization of features in PDPs through sequences of bit shifts. As future work, we intend to optimize the feature set for video streaming traffic by evaluating feature importance through algorithms such as Principal Component Analysis [Song et al. 2010] and implementing more sophisticated features. For example, the number of Payload Peak Points, which are points throughout the lifetime of a flow in which the payload size reaches a local maximum, seems to be relevant for video streaming classification [Dong et al. 2017]. Lastly, we intend to investigate the employment of more robust clustering algorithms with K-Flix, such as K-Medoids.

Acknowledgments

This work was financed in part by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001 and grant #88887.954253/2024-00, CNPq grants #311276/2021-0 and #405940/2022-0, and FAPESP grants #2020/05152-7, #2023/00673-7 and #2023/00764-2.

References

Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., Talayco, D., Vahdat, A., Varghese, G., and Walker, D. (2014). P4: programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95.

Bosshart, P., Gibb, G., Kim, H.-S., Varghese, G., McKeown, N., Izzard, M., Mujica, F., and Horowitz, M. (2013). Forwarding metamorphosis: fast programmable match-action processing in hardware for sdn. *SIGCOMM Comput. Commun. Rev.*, 43(4):99–110.

Boutaba, R., Salahuddin, M., Limam, N., Ayoubi, S., Shahriar, N., Estrada-Solano, F., and Caicedo, M. (2018). A comprehensive survey on machine learning for networking: Evolution, applications and research opportunities. *Journal of Internet Services and Applications*, 9.

Cannarozzo, L., Morais, T. B., de Souza, P. S. S., Gobatto, L. R., Lamb, I. P., Duarte, P. A. P. R., Azambuja, J. R. F., Lorenzon, A. F., Rossi, F. D., Cordeiro, W., and Luizelli, M. C. (2024). Spinner: Enabling in-network flow clustering entirely in a programmable data plane. In *NOMS 2024-2024 IEEE Network Operations and Management Symposium*, pages 1–9.

Cassola, A. (2025). K-Flix Repository. [Online] Available: *https://github.com/amauryTCassola/KFLIXP4*.

Coelho, B. and Schaeffer-Filho, A. (2022). Backorders: using random forests to detect ddos attacks in programmable data planes. In *Proceedings of the 5th International Workshop on P4 in Europe*, EuroP4 '22, page 1–7, New York, NY, USA. Association for Computing Machinery.

Dalmazo, B. L., Marques, J. A., Costa, L. R., Bonfim, M. S., Carvalho, R. N., da Silva, A. S., Fernandes, S., Bordim, J. L., Alchieri, E., Schaeffer-Filho, A., Paschoal Gaspary, L., and Cordeiro, W. (2021). A systematic review on distributed denial of service attack defense mechanisms in programmable networks. *International Journal of Network Management*, 31(6):e2163.

Dias Knob, L. A., Esteves, R. P., Granville, L. Z., and Rockenbach Tarouco, L. M. (2017). Mitigating elephant flows in sdn-based ixp networks. In *2017 IEEE Symposium on Computers and Communications (ISCC)*, pages 1352–1359.

Dong, Y.-n., Zhao, J.-j., and Jin, J. (2017). Novel feature selection and classification of internet video traffic based on a hierarchical scheme. *Comput. Netw.*, 119(C):102–111.

Friedman, R., Goaz, O., and Rottenstreich, O. (2021). Clustreams: Data plane clustering. In *Proceedings of the ACM SIGCOMM Symposium on SDN Research (SOSR)*, SOSR '21, page 101–107, New York, NY, USA. Association for Computing Machinery.

Hauser, F., Häberle, M., Merling, D., Lindner, S., Gurevich, V., Zeiger, F., Frank, R., and Menth, M. (2023). A survey on data plane programming with p4: Fundamentals, advances, and applied research. *Journal of Network and Computer Applications*, 212:103561.

Nguyen, T. T. and Armitage, G. (2008). A survey of techniques for internet traffic classification using machine learning. *IEEE Communications Surveys Tutorials*, 10(4):56–76.

Nunes, D. C., Coelho, B. L., Parizotto, R., and Schaeffer-Filho, A. (2023). Serene: Handling the effects of stragglers in in-network machine learning aggregation. In *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*, pages 1–10.

Parizotto, R., Coelho, B. L., Nunes, D. C., Haque, I., and Schaeffer-Filho, A. (2023). Offloading machine learning to programmable data planes: A systematic survey. *ACM Comput. Surv.*, 56(1).

Sandvine, I. (2023). Global internet phenomena report 2023. [Online]. Available: *https://www.sandvine.com/global-internet-phenomena-report-2023*.

Shamsimukhametov, D., Liubogoshchev, M., Khorov, E., and Akyildiz, I. F. (2021). Youtube, netflix, web dataset for encrypted traffic classification. [Online]. Available: *https://dx.doi.org/10.21227/s7x7-wd58*.

Song, F., Guo, Z., and Mei, D. (2010). Feature selection using principal component analysis. In *2010 International Conference on System Science, Engineering Design and Manufacturing Informatization*, volume 1, pages 27–30.

Usama, M., Qadir, J., Raza, A., Arif, H., Yau, K.-l. A., Elkhatib, Y., Hussain, A., and Al-Fuqaha, A. (2019). Unsupervised machine learning for networking: Techniques, applications and research challenges. *IEEE Access*, 7:65579–65615.

Xavier, B. M., Silva Guimarães, R., Comarela, G., and Martinello, M. (2022). Map4: A pragmatic framework for in-network machine learning traffic classification. *IEEE Transactions on Network and Service Management*, 19(4):4176–4188.

Xiong, Z. and Zilberman, N. (2019). Do switches dream of machine learning? toward in-network classification. In *Proceedings of the 18th ACM Workshop on Hot Topics in Networks*, HotNets '19, page 25–33, New York, NY, USA. Association for Computing Machinery.

Zhang, J., Xiang, Y., Wang, Y., Zhou, W., Xiang, Y., and Guan, Y. (2013). Network traffic classification using correlation information. *IEEE Transactions on Parallel and Distributed Systems*, 24(1):104–117.

Zhang, Q. and Sun, S. (2010). A centroid k-nearest neighbor method. In Cao, L., Feng, Y., and Zhong, J., editors, *Advanced Data Mining and Applications*, pages 278–285, Berlin, Heidelberg. Springer Berlin Heidelberg.

Zhu, X. and Zhang, Y. (2022). Machine-learning-assisted traffic classification of user activities at programmable data plane. In *2022 23rd Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pages 01–04.