# Achieving Robust Performance for Traffic Classification Using Ensemble Learning in SDN Networks

Ting Yang, Serdar Vural, Peng Qian, Yogaratnam Rahulan, Ning Wang, Rahim Tafazolli

5G & 6G Innovation Centre, Institute for Communication Systems,
University of Surrey, Guildford, Surrey, UK
e-mail: {ting.yang, s.vural, peng.qian, y.rahulan, n.wang, r.tafazolli}@surrey.ac.uk

ICC 2021 - IEEE International Conference on Communications | 978-1-7281-7122-7/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICC42927.2021.9500571

***Abstract*—Software-defined networking (SDN) enables centralized control of a network of programmable switches by dynamically updating flow rules. This paves the way for dynamic and autonomous control of the network. In order to be able to apply a suitable set of policies to the correct set of traffic flows, SDN needs input from traffic classification mechanisms. Today, there is a variety of classification algorithms in machine learning. However, recent studies have found that using an arbitrary algorithm does not necessarily provide the best classification outcome on a dataset, and therefore a framework called *ensemble* which combines individual algorithms to improve classification results has gained attraction. In this paper, we propose the application of the ensemble algorithm as a machine learning preprocessing tool, which classifies ingress network traffic for SDN to pick the right set of traffic policies. Performance evaluation results show that this ensemble classifier can achieve robust performance in all tested traffic types.**

## I. Introduction

The $5^{th}$ Generation (5G) networks are expected to support a variety of services for a multitude of industry verticals that have a diverse set of performance and service requirements [1]. To support various use cases, the 5G network architecture is designed as a service-based one, following a micro-services structure, and enabling multiple virtual networks running in parallel, each configured for a different use case. Such virtual networks are referred to as *network slices*, which need to support suitable policies, and where necessary, be configured dynamically to adapt to changing network conditions and meet their user-case requirements.

Dynamic establishment and life-cycle management of network slices are possible when networks are flexible and programmable. This can be achieved by software-defined networking (SDN) and network functions virtualization (NFV), to support a multitude of diverse end-to-end services over a common underlying physical infrastructure [2]. The key idea of SDN is to make networks programmable by separating the control and data planes of networking equipment. In an SDN network, an SDN controller determines packet forwarding rules and then programs network elements accordingly [3].

The complexity, variety and number of traffic flows will be higher in 5G, especially with the wide adoption of virtual network functions and dynamic deployment of virtual network slices, the cost of operating a highly flexible and programmable network can become substantial [4]. Some of these traffic types have stringent requirements on delay, which necessitates minimal processing delay while handling this variety and volume of traffic. Therefore, it is essential to first accurately classify 5G traffic flows, and then apply these rules in networking devices. This can be achieved with an automated SDN infrastructure [5] which can adapt to changing traffic patterns, and meet the service level agreements (SLA) of a multitude of traffic types, whilst best using the network's physical resources.

Although, as of today, there is a large body of research on SDN solutions that can perform common networking tasks, automation of SDN networks is still in its infancy. By collecting telemetry information, machine learning (ML) algorithms can learn and predict traffic patterns, and then help SDN to better provision the physical network. In this paper, our focus is on traffic classification in an SDN network. Different ML algorithms have been first applied to datasets for different application traffic types to evaluate algorithm classification performance. To improve classification accuracy, we argue that a classifier should have two processing layers when selecting the most suitable ML strategy for a given traffic type. As such, we propose to apply the ensemble ML framework [6] to the specific problem of network traffic classification as an input to SDN. Promising results have been obtained, demonstrating the advantage of this approach in both classification accuracy and performance robustness.

The paper is organized as follows. In Section II, we present the related work. In Section III, some preliminary information on the classification framework is provided. Then, in Section IV, we propose the ensemble classifier for traffic classification in SDN networks. Section V presents the performance results. Finally, in Section VI, concluding remarks are made and future work directions are provided.

## II. Related Work

With the advances in computer hardware and software technologies, ML has become realistically practical, and applicable to many science and technology fields in recent years,

978-1-7281-7122-7/21/$31.00 ©2021 IEEE

making it a popular approach to address today's various open technical problems. One such area is traffic classification in communications networks; with a multitude of application scenarios and a diverse set of networking and connectivity solutions, traffic engineering flows in today's networks have become highly complex. As SDN is now getting increasingly popular due to its benefits in network programmability, there has been recent interest in applying ML solutions to SDN in order to achieve different goals, one of which is traffic classification.

Identifying the type of traffic flows is necessary for effective operation of networks. For instance, elephant flows can easily lead to network congestion if not properly managed. However, detecting a specific traffic type within a blend of different traffic flow types in an operational network is challenging. To address this issue, in [7], a machine learning algorithm, namely C4.5, is proposed specifically for identifying real-time elephant flows in data centers. The authors point out that having more data features with C4.5 can improve classification performance, but leads to a longer response time for real-time traffic.

In addition to identifying only one type of traffic flow, machine learning algorithms can also be used to perform classification on a finer level of granularity in order to detect multiple types of flows at once. In [8], authors propose a platform to classify different application traffic streams. A decision tree machine learning model called c5.0 is trained to detect the top forty most popular applications in Google Play store. Eight out of forty application types can be accurately detected, yet for the rest, relatively poor results are obtained. In [9], authors use deep packet inspection and a semi-supervised machine learning algorithm, called Laplacian support vector machine for traffic classification. The algorithm extracts sixty features from the tested traffic flows, nine of which are adopted by leveraging the complexity of the classification model.

Training only a single classification model may only work well for specific types of flows but may not perform well when multiple types of traffic co-exist. As such, recent studies propose *ensemble* algorithms which can combine multiple classifiers to achieve better classification performance. The study presented in [6] targets at classifying multiple traffic flows using an ensemble approach. Another application of ensemble algorithms is detection of network traffic anomaly. Authors in [10] compare various ensemble frameworks for detecting anomalies in network traffic, whereas in [11], the ensemble framework is further studied in an SDN environment for detecting DDoS traffic.

The study presented in [6] adopts a multi-class traffic classification structure, where a pipeline of classification models are applied to incoming traffic flows. In pipeline-based multi-classifier approaches, each model in the pipeline predicts a different type of traffic. When a traffic flow arrives, it is first passed to the first classification model which represents a certain type of traffic. If not matched, then the traffic is forwarded to the next model. This process continues until a model in the pipeline can achieve sufficient detection performance and determine the correct traffic type.

Although an effective approach in classifying a set of a few traffic flows types, pipeline based multi-model classifiers are not suitable when a large number of traffic types are to be handled by the ML system. This is because multiple models in the pipeline may have satisfactory results for a given received traffic type, even though they may have different classification performance. Since the traffic leaves the pipeline when a model determines the traffic type, a pipeline based multi-model classifier cannot guarantee that the best model is used for classification. Hence, the ordering of the models in the pipeline impacts the final classification results. The best pipeline configuration can be picked for a given set of traffic types, yet this configuration may be different for a different set. The cost of deciding the optimal configuration for a wide variety of traffic types can be exponential. Furthermore, in the worst case scenario, a traffic flow has to pass all the models before reaching the correct traffic type. When more traffic types need to be classified, the pipeline must also grow with new models added to achieve better classification performance.

In summary, there is still a gap in achieving the best classification performance for multiple types of traffic, whilst also keeping the ML system scalable and efficient. With this motivation, we propose an ensemble framework which combines results from multiple classification models in a two-tier architecture. In the following sections, we present this framework.

## III. Classification Framework

Before presenting the proposed application of the ensemble algorithm to traffic classification in SDN networks, in this section, we first introduce some underlying concepts necessary to build a classification framework. We first explain how data features are extracted in an SDN infrastructure, and then outline how traffic policies are derived by the SDN controller with the use of an ML Classifier.

### A. Feature Extraction and Engineering

Machine learning algorithms rely on accurate extraction of key data characteristics, referred as *features* to be able to perform high-grade data classification. When applying ML to network traffic, traffic traces are needed to be collected by the Internet service providers or network operators for a certain period of time, which can then be fed into an ML algorithm as a dataset. A traffic trace can be collected for a specific traffic flow type or a mixture of multiple traffic flow types. Once traffic traces have been collected, classification algorithms can be trained to extract statistical features, e.g. average packet length, average payload size, packet inter-arrival time, and minimum/maximum TCP window size [6]. In general, having a fairly large number of available features provides flexibility to an ML algorithm, which can then select the most relevant set of features that provide the best classification accuracy.

The success of machine learning and what it can provide depends on what data are available to ML, i.e. data characteristics. Therefore, it is necessary to identify what sort of data

SDN can provide to ML. This can then help us understand the extra information that an ML algorithm can provide to SDN in return. In this section, we briefly outline the main technical challenge: the data.

In an SDN-controlled network, the role of the SDN controller is to classify received "unknown" packets by reading packet headers and then to insert an appropriate policy rule in the network infrastructure[1] (switches, routers, etc.) so that similar packets can later be directly served by this new traffic policy rule in hardware without the intervention of the controller. This is illustrated in Figure 1.

As it can be seen in Figure 1, when an unknown packet arrives at the SDN switch, the switch forwards this packet to the SDN controller, where a suitable policy is searched, in order to be applied to the packet based on its packet header fields. If such a policy is found, then a policy rule is added to the switch. The policy is later directly applied to similar packets without the need for the controller to make another decision. This operation provides the capability to dynamically handle emerging traffic flows, and apply suitable traffic policies. When needed, the controller can then modify the policy rules to alter the traffic treatment applied to packets that belong to traffic flows, which is the programmability benefit of SDN.

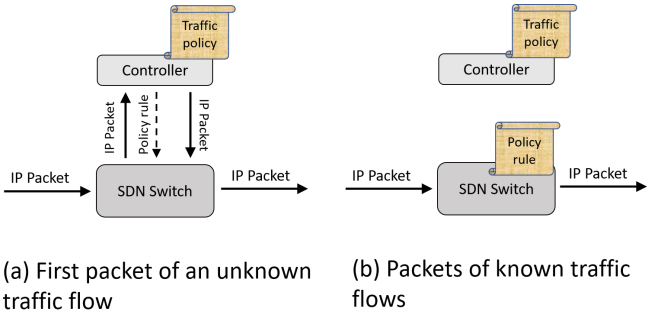


Fig. 1: Applying an ML policy to received IP packets.

Table I presents a sample set of *match fields* that define a typical traffic flow in an SDN network. SDN switches read packet headers and check whether a set of match fields apply to each packet, to determine whether any suitable policy rule can be applied.

TABLE I: Match fields of a typical traffic flow in an SDN network [12], presented to ML algorithms as data attributes.

| Feature | Description |
|---|---|
| SrcMAC | Source MAC Address |
| DstMAC | Destination MAC Address |
| SrcIP | Source IP Address |
| DstIP | Destination IP Address |
| SrcPort | Source Port |
| DstPort | Destination Port |
| VLANID | VLAN ID |

[1]To program hardware switches, the controller exchanges some control messages with the hardware equipment. The most common SDN protocol in use today delivering these control messages is OpenFlow [12], which is the defacto SDN protocol in use today in research and production networks. OpenFlow specifications are provided by the Open Networking Foundation [13].

It must be noted that, in addition to these readily available features listed in Table I, when unknown incoming flows arrive at the SDN controller, the controller can also choose to buffer the first $N$ packets to derive statistical information for each feature (e.g. average packet size). However, buffering the first $N$ packets to obtain additional features comes at the cost of introducing extra cost and processing delay for these traffic flows [14], which may not be feasible for delay-sensitive applications such as Ultra Reliable Low Latency Communications (URLLC) traffic in 5G networks. In summary, to apply an ML algorithm to an SDN network, the algorithm must be able to utilize whatever SDN can read practically, i.e. packet header fields, and consider the best possible set of features for each traffic type.

Once the data features have been obtained, classification algorithms typically compute distance metrics, using the data points, in order to obtain a quantitative measure to identify the similarity between two separate data points, where each data point is defined by a value-set of the collected features. However, some features may not be numerical, e.g. IP addresses. Such metrics are mapped to a finite numerical space, before making distance computations.

### B. Derivation of Traffic Policies

Traditionally, traffic engineering policies in switching hardware are applied at the network planning stage, since network operators would have a priori/expected knowledge of the network traffic. However, this static solution no longer meets the requirements imposed by next generation networks, where real-time deployment of services and network slices is expected to generate a highly dynamic network.

With the help of SDN, operators now have the capability to deploy traffic policy rules dynamically, based on emerging traffic patterns. However, a traffic classification module is essential for SDN for two main reasons. First, SDN requires prior knowledge on different traffic types and packet characteristics of each traffic type that is expected to be serviced by the network, so that suitable policy rules can be selected and applied. Secondly, as switching hardware has a finite and limited flow space to accommodate traffic flow rules [15], it is impossible to store individual rules for every traffic flow; in fact a single rule per traffic class is sufficient. Therefore, an ML classifier that has the ability to dynamically classify a traffic flow into a specific category is needed.

We consider a supervised multi-class classifier, which is trained with a labeled training dataset, and derives traffic policies that are then provided to the SDN Controller. This is illustrated in Figure 2.

It must be noted that, as mentioned in Section III-A, an SDN controller can only define its data features based on packet header fields which it has access to, unless an additional metadata source exists. The ML classifier uses all or a subset of packet header fields as data features, and derives a traffic policy. This policy essentially defines the following:

1) The set and value of packet header fields, which are used to classify the packet as a certain traffic type,

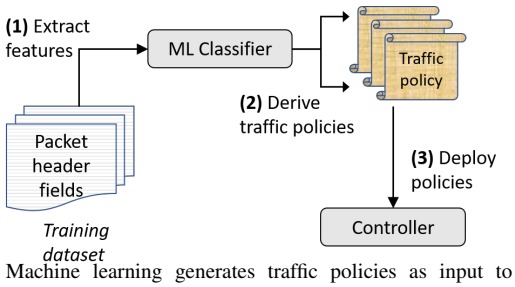

Fig. 2: Machine learning generates traffic policies as input to an SDN controller.

2) The set of actions to be taken to apply a certain level of QoS treatment to the packet based on this traffic type, in order to meet the QoS requirements of the traffic type.

Once the traffic policy is determined it is deployed at the SDN controller to be used to classify received packets.

## IV. The Ensemble ML Classifier

When defining a classifier for network traffic traces, the typical approach followed in existing studies is either to optimize a specific machine learning algorithm or to apply a number of algorithms and compare their performances in order to determine the most suitable one. However, a system that relies only on one machine learning algorithm is not robust. Therefore, we follow a different approach and adopt the ensemble learning framework to train multiple classifiers at once to obtain a final prediction by jointly considering the predictions of multiple trained classifiers.

Figure 3 shows the workflow of the proposed two-tier ensemble approach. On the first tier, different classifiers are trained independently on the same training set. This training process is no different from when a second-tier is not in use (i.e. no ensemble). The difference lies in the second tier; rather than picking one of these multiple trained classifiers, the ensemble algorithm focuses on how to use the prediction results of the first tier to improve the final prediction results.

In our approach, the predictions of the first tier are combined as the training dataset for the classifier at the second tier to predict the final results. This way of building a classification model aims to combine several models to produce a powerful *ensemble classifier*. The ensemble classifier is flexible as it allows addition of new machine learning models at the first-tier.

Algorithm 1 shows the detailed procedure for building the proposed ensemble classifier. The traffic flow dataset is labeled, i.e. it includes traffic type labels of the data samples indicating what type of traffic each sample belongs to. The datasets is split into a training dataset and a test dataset, which are denoted as $D_{train}$ and $D_{test}$, respectively. The ensemble procedure (see line 1) also accepts the necessary set of parameters for the single classifiers, denoted by $P$.

The base-level (i.e. the first tier) procedure (line 2) accepts $D_{train}$ and the set of classifiers as its parameters. The training data $D_{train}$ is then further split into training and test datasets which are denoted as $X_{train}$ and $X_{test}$, respectively. A 5-fold cross-validation is used, where the first-tier classifiers are

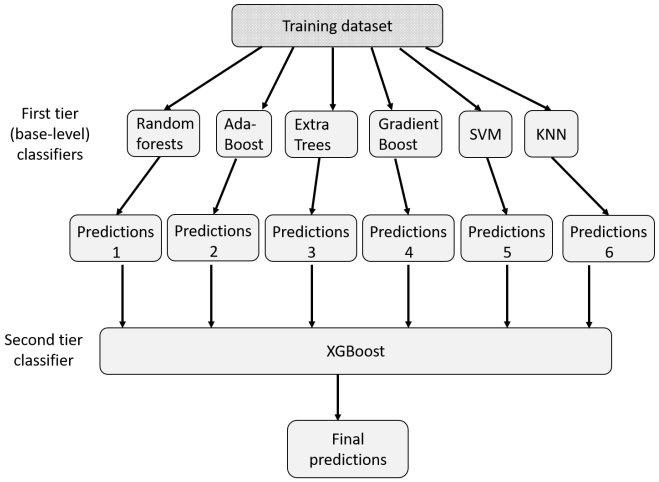

Fig. 3: Workflow of the ensemble classifier.

**Algorithm 1** Ensemble Traffic Classification Algorithm

**Input:** $N$ base-level classifiers, $D_{train}$, $D_{test}$
**Output:** Predicted traffic type of an unknown flow

1: **procedure** ENSEMBLE($D_{train}, P$)
2: **procedure** BASE-LEVEL($D_{train}$, classifiers)
3: Split $D_{train}$ to $X_{train}$ and $X_{test}$
4: **for** $i \leftarrow 1, n$ **do**
5: $trainedClassifier$
6: $\leftarrow classifier_i(X_{train}, P)$
7: $prediction_i \leftarrow trainedClassifier(X_{test})$
8: $predictions[i] \leftarrow prediction_i$
9: **end for**
10: **end procedure**
11: **procedure** SECOND-LEVEL($predictions, P$)
12: $trainedClassifier$
13: $\leftarrow XGBoost(predictions, P)$
14: **end procedure**
15: **procedure** TRAFFIC-CLASSIFIER($traffic_{flow}$)
16: $type_i \leftarrow Ensemble(traffic_{flow})$
17: **end procedure**
18: **end procedure**

the six most widely used learning models available on scikit-learn [16]: KNN, support vector machine (SVM), random forest, AdaBoost, extra trees, and gradient boosting. These six models are trained, and their predicted results are stored in an array denoted by $predictions$. We train the six models on the same computer sequentially (presented by a for loop in the algorithm, as in line 4). In practice, to speed up the training process, these models can be trained in parallel on separate machines.

The output of the first-tier classifiers is the predicted traffic types of the six individual machine learning models, denoted by $predictions$. This is then fed into the second-tier classifier as training data in order to train it (see line 11). At the second tier, we choose the famous boosted tree learning model, called XGBoost [17], for predicting the final results. XGBoost was

built to optimize large-scale boosted tree algorithms, and is suitable for large datasets. A trained XGBoost model automatically calculates the *feature importance* of the predictive model. In other words, using XGBoost at the second tier, we can use the feature importance evaluation results as an indication of the relative contribution of each individual classifier of the first tier to the final prediction results. We can then either deploy all the trained first-tier classifiers or choose the ones that contribute the most to the final prediction results.

Finally, line 15 in Algorithm 1 shows the main method that calls the ensemble classifier to determine the type of a given traffic flow.

## V. Performance Evaluation

In this section, the performance results of the proposed ensemble classifier are presented when applied as a traffic classification engine in SDN networks.

### A. Experiment Setup

The dataset used in performance tests includes traffic trace data collected with WireShark [18], for five types of applications where each application represents a traffic type, as provided in [19]:

- Video: YouTube
- Voice: Google VoIP
- Bulk data transfer: FTP
- Music: Google Music
- Interactive data: Google Text Chat

We use the Python Scapy [20] module to clean the collected files, and extract the various features of the traffic flows, such as the traditional 5-tuples, i.e. source IP address, destination IP address, source port number, destination port number, and protocol type. Scapy can also dissect the packet to get its header and payload sizes. We use source and destination IP addresses and port numbers, as well as the packet payload size as our classification features.

The dataset contains $10^5$ traffic flows, where each traffic type constitutes an equal proportion of the total set of flows. We use a split of $75\%$ and $25\%$ for the training and test data sets, respectively. The training sets are then used to train machine learning classifiers.

The key performance metrics[2] used in performance results are provided for clarity of presentation, as in Table II.

### B. Experiment Results

As introduced in Section IV, for the first tier of the ensemble classifier, we use SVM, random forest, extra trees, AdaBoost, and gradient boost classifier algorithms, which are the most common and popular ML algorithm for traffic classification. In the second tier, we utilize the XGBoost classifier for generating the final prediction results. We compare the performance of the proposed ensemble classifier against those obtained by directly using individual single classifiers (e.g. use random forest tree only). We also compare against a *max-voting framework* in which the second layer predictions are based on max voting results, i.e. the traffic class that gains the most votes by the first tier prediction results is chosen. The max-voting framework's first layer is exactly the same as the ensemble classifier.

[2]These are commonly used metrics in machine learning.

TABLE II: Definition of accuracy, recall and precision.

| Metric | Definition | Interpretation |
|---|---|---|
| Accuracy | The number of correctly predicted data points out of all the data points. | Accuracy measures the percentage of the total correctly predicted traffic flows out of all the traffic flows. |
| Recall | The fraction of relevant instances among the retrieved instances. | Recall of a specific traffic class measures the fraction of the correctly predicted traffic flows of this class among all the traffic flows of this class. |
| Precision | The fraction of the total amount of relevant instances that were actually retrieved | Precision of a specific traffic class measures the fraction of the correctly predicted traffic flows of this class among all the traffic flows predicted as this class. |

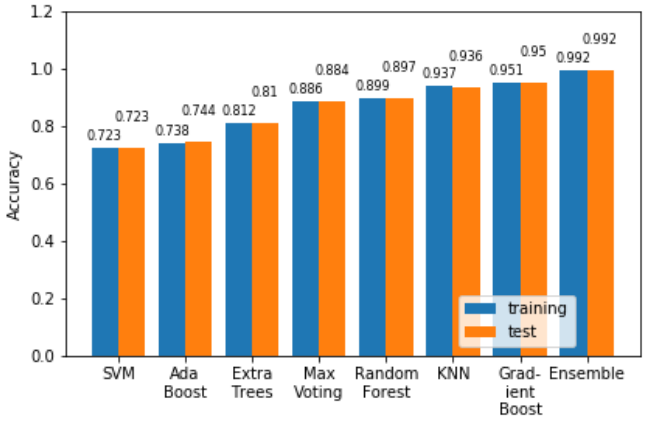


Fig. 4: Accuracy of different ML algorithms.

For evaluation purposes, we apply a $K$-fold cross-validation process, where $K = 5$ in our experiments presented in this section, which means the process is repeated five times to shuffle the data and avoid data over-fitting. Furthermore, the dataset is scaled using the MinMaxScaler implemented in scikit-learn [21], which is an important pre-processing procedure for SVM and KNN. Scaling is not necessary for decision-tree based algorithm, hence omitting this scaling process can be advantageous for real-time classification in an SDN network.

Figure 4 shows the overall multi-class classification accuracy of various machine learning algorithms. It can be observed that different machine learning algorithms can have very different accuracy results. It is worth noting that the SVM classifier has a relatively poor performance. This is because, in SDN networks, only a limited number of features can be used for classification, while SVM normally works well with a small dataset size and a large number of features. In

TABLE III: Recall and precision performance of classifiers for individual traffic types.

| | | Single classifiers | | | | | | Two-tier classifiers | |
|---|---|---|---|---|---|---|---|---|---|
| **Traffic Types** | **KPI** | KNN | SVM | RandomForest | ExtraTrees | AdaBoost | GradientBoost | MaxVoting | Ensemble |
| GoogleChat | Recall | 0.965 | 0.973 | 0.973 | 0.973 | 0.975 | 0.977 | 0.975 | 0.998 |
| | Precision | 0.966 | 0.788 | 0.981 | 0.779 | 0.985 | 1.0 | 0.965 | 1.0 |
| FileTransfer | Recall | 0.965 | **0.165** | 0.897 | **0.549** | **0.313** | 0.963 | 0.806 | 0.990 |
| | Precision | 0.871 | **0.592** | 0.799 | 0.841 | **0.601** | 0.871 | 0.815 | 0.981 |
| Music | Recall | 0.933 | 0.825 | 0.900 | 0.853 | 0.885 | 0.959 | 0.912 | 0.993 |
| | Precision | 0.962 | 0.669 | 0.853 | 0.718 | 0.771 | 0.961 | 0.815 | 0.986 |
| VoIP | Recall | 0.851 | 0.804 | 0.871 | 0.828 | 0.776 | 0.871 | 0.877 | 0.976 |
| | Precision | 0.903 | 0.910 | 0.896 | 0.927 | 0.767 | 0.926 | 0.912 | 0.986 |
| YouTube | Recall | 0.969 | 0.842 | 0.845 | 0.844 | 0.765 | 0.981 | 0.848 | 0.996 |
| | Precision | 0.986 | 0.619 | 0.984 | 0.834 | 0.578 | 1.0 | 0.975 | 1.0 |

contrast, the proposed ensemble classifier achieves the best accuracy results by mitigating the impact of the limited number of features. Furthermore, the ensemble classifier exhibits satisfactory generalization performance, i.e. the over-fitting problem is mitigated as shown in Figure 4. This is important in practical networks since high network dynamicity can exhibit very different traffic patterns. Overall, the performance results in Figure 4 suggest that for different classification applications, using the proposed ensemble classifier yields better results than relying only on a single classifier.

Table III illustrates the "recall and precision results" in terms of individual types of traffic; The closer these values are to a value of 1, the better. It can be observed that, overall, each classifier has its strength and weakness in classifying certain types of traffic flows. For instance, SVM, ExtraTrees and AdaBoost classifiers perform extremely poorly on the "file transfer" type traffic flows. In contrast, using the proposed ensemble classifier, we achieve consistently high performance for all different traffic types.

In summary, the results presented in Figure 4 and Table III show that the proposed ensemble classifier not only performs well on the whole dataset but also works well for individual traffic types.

## VI. Conclusion

In this paper, we propose an ensemble classifier and evaluate its performance in traffic classification in SDN networks. Since SDN can read packet header fields only, the number of data features available to classification algorithms in real-time is limited. Instead of relying on various statistical features of traffic traces for training the model, we use the easily collected destination and source port numbers for training the classifier. To compensate for the lack of classification features, we propose a two-tier classification workflow that combines the advantages of multiple classifiers. Experimental results show that the proposed ensemble classifier not only achieves an overall high classification accuracy, but also improves the classification performance for individual traffic types. Future research will test the ensemble classifier for imbalanced datasets, and evaluate its performance with semi-supervised ML models.

## Acknowledgment

This work is partially funded by EPSRC NG-CDI project (EP/R004935/1). The authors would also like to acknowledge the support of the University of Surrey's 5G Innovation Centre (5GIC http://www.surrey.ac.uk/5gic) members for this work.

## References

[1] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina, "Network Slicing in 5G: Survey and Challenges," *IEEE Commun. Mag.*, 2017.
[2] J. Xie, F. R. Yu, T. Huang, R. Xie, J. Liu, C. Wang, and Y. Liu, "A Survey of Machine Learning Techniques Applied to Software Defined Networking (SDN): Research Issues and Challenges," *IEEE Commun. Surv.*, 2019.
[3] B. A. A. Nunes, M. Mendonca, X. Nguyen, K. Obraczka, and T. Turletti, "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks," *IEEE Commun. Surv.*, 2014.
[4] Ericsson, "Scalable network opportunities: An economic study of 5G network slicing for IoT service deployment," 2018.
[5] M. Wang, Y. Cui, X. Wang, S. Xiao, and J. Jiang, "Machine learning for networking: Workflow, advances and opportunities," *IEEE Network*, 2018.
[6] S. E. Gómez, B. C. Martínez, A. J. Sánchez-Esguevillas, and L. H. Callejo, "Ensemble network traffic classification: Algorithm comparison and novel ensemble scheme proposal," *Computer Networks*, 2017.
[7] P. Xiao, W. Qu, H. Qi, Y. Xu, and Z. Li, "An efficient elephant flow detection with cost-sensitive in SDN," in *INISCom*, 2015.
[8] Z. A. Qazi, J. Lee, T. Jin, G. Bellala, M. Arndt, and G. Noubir, "Application-Awareness in SDN," in *ACM SIGCOMM*, 2013.
[9] P. Wang, S. Lin, and M. Luo, "A Framework for QoS-aware Traffic Classification Using Semi-supervised Machine Learning in SDNs," in *IEEE SCC*, 2016.
[10] I. P. Possebon, A. S. Silva, L. Z. Granville, A. Schaeffer-Filho, and A. Marnerides, "Improved Network Traffic Classification Using Ensemble Learning," in *IEEE ISCC*, 2019.
[11] V. Deepa, K. M. Sudar, and P. Deepalakshmi, "Design of Ensemble Learning Methods for DDoS Detection in SDN Environment," in *ViTE-CoN*, 2019.
[12] [Online]. Available: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.3.5.pdf
[13] [Online]. Available: https://www.opennetworking.org/
[14] W. Li and A. W. Moore, "A Machine Learning Approach for Efficient Traffic Classification," in *IEEE MASCOTS*, 2007.
[15] X. Wang, Q. Deng, J. Ren, M. Malboubi, S. Wang, S. Xu, and C. Chuah, "The Joint Optimization of Online Traffic Matrix Measurement and Traffic Engineering For Software-Defined Networks," *IEEE/ACM TON*, 2020.
[16] [Online]. Available: https://scikit-learn.org/stable/
[17] [Online]. Available: https://xgboost.readthedocs.io/en/latest/
[18] [Online]. Available: https://www.wireshark.org/
[19] V. Tong, H. A. Tran, S. Souihi, and A. Mellouk, "A Novel QUIC Traffic Classifier Based on Convolutional Neural Networks," in *IEEE GLOBECOM*, 2018.
[20] [Online]. Available: https://scapy.net/
[21] [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html