An Intelligent Traffic Classification based optimized routing in SDN-IoT: A Machine Learning Approach

Isaac Ampratwum

Thesis submitted to the University of Ottawa in partial Fulfillment of the requirements
for the Master of Applied Science degree in
Electrical and Computer Engineering

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Abstract

Due to speedy increase in IoT devices and its QoS requirements, providing networks solutions to meet this demand has become a major research issue. Providing fast and reliable routing paths based on the QoS requirement of IoT device is very vital. Software defined networking is one of the most current interesting development in the field of research. A new paradigm, SDN-IoT, leveraging the advantages of SDN architecture on IoT networks have been proposed to improve network quality. Also, application of artificial intelligence (AI) in SDN for traffic engineering is widely researched. In this work, we first propose a machine learning based traffic load classification into the traffic's QoS requirements. Then, a deep learning route optimization model based on the traffic classification is proposed. The model chooses the route that meets the QoS demands like latency of the identified traffic. The simulation results show that our proposed solutions perform very well and better than some significant works in the same area.

Acknowledgments

First, I would like to express my sincere gratitude to God for his guidance and help throughout my period of study. Secondly, a big thank you to Prof. Amiya Nayak for his continuous support, motivation, advice and immense knowledge throughout my thesis work. I could not have imagined having a better mentor for my master's study. Also, I would like to thank my family and friends for providing me with unfailing support and encouragement throughout the process of researching and writing my thesis. This accomplishment would not have been possible without them. Thank you.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

IoT	Internet of Things
QoS	Quality of Service
SDN	Software Defined Networking
ML	Machine Learning
API	Application Programming Interface
NN	Neural Network
ANN	Artificial Neural Network
DNN	Deep Neural Network
VNF	Virtual Networks Function
SHAP	Shapely Adaptive Explanations
SFS	Sequential Forward Selector

1.0. Introduction

1.1. Motivation

Internet of Things (IoT) refer to uniquely identifiable objects (things) and their virtual representations in an internet-like structure [1]. The number of Internet of Things (IoT) devices is projected to grow to 20 billion in 2020 [2]. As the number of IoT devices increase exponentially, traffic volume on the global network increases. Owing to the diversity in IoT devices such as smart phones, smart cars, cameras, thermostat, bulb etc., real world deployment IoT is highly heterogeneous and complex.

IoT devices produce large traffic volumes and diverse traffic types with different QoS requirements. For Example, certain IoT devices like CCTV cameras require large bandwidth and low latency to keep video streams flowing whereas Voice over IP (VoIP) applications are delay-sensitive but require low bandwidth. Effective routing is key in meeting the diverse QoS requirements for IoT devices. Routing methods employed in traditional networks are mainly distributed. Though these routing protocols may be resilient, they are less flexible and can be difficult to manage in large networks.[3]

Due to the various limitations of traditional networks such as flexibility, manageability and extensibility [4], IoT traffic will be a challenge. Different approaches are being investigated to handle these complex and heterogeneous IoT networks. One of the promising frameworks for IoT networks is Software Defined Networking (SDN). Also providing intelligence in the network can help solve with some of these issues.

Software Defined Networking is a trending paradigm adopted in IoT networks to provide effective traffic management and required Quality of Service (QoS). IoT network operating on SDN framework can be referred to as SDN-IoT. SDN decouples the switches into 3 layers, data layer, control layer and application layer. The network is overseen by a centralized entity called the controller. The Controller has a global view of the network by monitoring and sampling real-time information from the network nodes and packets. The controller is responsible for making routing decisions for the network and can program the

network dynamically. The centralized nature of the SDN architecture, it's programmability and the controller's ability to collect real-time data makes it possible to apply some 'intelligence' (Machine learning techniques) for effective routing and QoS provisioning [5]. Machine learning (ML) is a field of Artificial intelligence and can be defined as the scientific study of algorithms and statistical models that computer systems use to perform a specific task without using explicit instructions, relying on patterns and inference instead[12]. Leveraging the programmability of SDN, we can use ML to provide QoS solutions for real time use.

Our work will focus on providing an SDN-IoT design that uses ML to provide QoS routing. We will classify the traffic and use the QoS requirements of the traffic to provide an optimal routing solution. Several works have been conducted in the domain of traffic classification and routing optimization. Most research works have treated traffic classification and routing optimization in isolation [21] [22] [23], but we will be providing a solution that combines both approaches for traffic engineering. Existing classification techniques like port-based [20] has become ineffective as most applications are using varying ports lately. DPI on the other hand achieves excellent results but is computationally costly. ML can provide efficient classification results at very low computation cost. Majority of ML-based traffic classification works done focus on identifying the corresponding application [23] [24] [25]. With increasing number of applications, this approach is impractical. On the other hand, existing routing algorithms like OSPF is a best effort routing protocol. This implies that when the network is congested, packets cannot be rerouted hence failing to meet the desired QoS of packets when network traffic is high.

To address the stated challenges, we will be applying ML algorithms is SDN-IoT network for traffic classification based optimized routing.

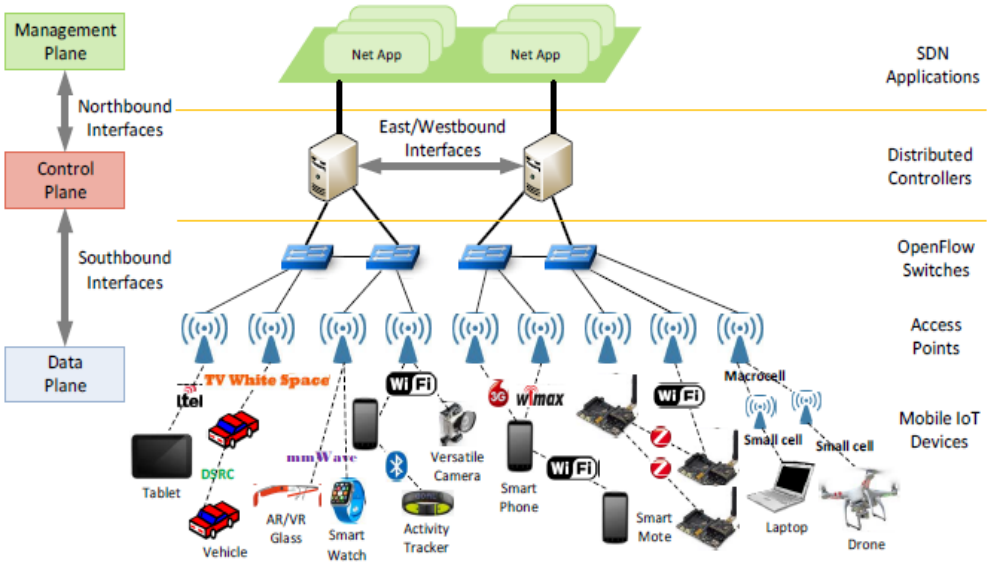

Figure 1: SDN-IoT architecture [6]

1.2. Objectives

Given the flexibility offered by SDN, the goal of this work is to investigate how to apply Machine Learning techniques to SDN-IoT networks to achieve QoS provisioning. We will look at using ML to classify the network traffic based on individual QoS requirements and predict a route that achieves the QoS requirements.

We will aim at proposing machine learning solutions that are lightweight and hence can be deployed easily in real life IoT environments.

1.3. Contribution

- We evaluate multiple ML algorithms on a chosen dataset and propose a model that will classify network traffic data into various QoS requirements. Our model will use statistical features of the packet to avoid packet inspection and to also provide fast classification.
- We also investigate the optimal set of statistical features that can provide good classification within a short time.

- We finally propose an intelligent routing approach based on the QoS requirement of the classified traffic. We employ supervised learning techniques in both contributions.

1.4. Thesis Organization

The thesis is organized as follows:

- Chapter 2 describes the concept of SDN and Machine Learning. We give an overview of existing and ongoing related works in traffic classification and route optimization. We point out the contributions and the downsides of these works.
- Chapter 3 describes our QoS aware traffic classification approach. Our proposed solution is explained in detail. The implementation procedure and results are presented.
- Chapter 4 presents our second contribution which is route optimization based on the classified traffic. We outline the approach, implementation and the achieved results.
- Chapter 5 gives a summary of the work done and discusses further work that can be done to improve upon our work.

2.0. Background and Related Work

This research utilizes machine learning to classify traffic and predict QoS based routes in SDN-IoT networks. In this session, we give an overview of the concept of SDN and Machine Learning. We also present a review on works on Machine learning based traffic classification and routing in software defined networks.

2.1. Software Defined Networking

The internet as we know it is mainly composed of interconnection of traditional or legacy networks. Legacy networks consist of proprietary network devices such as switches and routers responsible for both making routing decisions and forwarding data. These network devices perform traffic routing based on partial knowledge on the network. The Legacy network structure is distributed. Network devices are made up of several thousands of codes that are proprietary without any room for flexibility. One of the major limitations of traditional networks is the complexity. Over the years, network requirements have evolved. As these requirements evolve, the more complex the design of network devices and the higher the cost of these devices. It is also difficult to implement new policies in traditional networks. Adding a new device or implementing a new service in an existing large network would require a rigorous configuration spanning several network devices. This requires a lot of time and financial resources. Traditional networks are clearly lacking when it comes to meeting current network requirements especially with the rapid increase in the use of IoT devices.

Software defined networking has gained a lot of attraction in recent years. A lot of research has been done in recent years to see how we can utilize the flexibility offered by SDN to manage traffic and improve computer communications especially with the rising use of IoT devices. SDN is simply decoupling the network switch into layers. The intelligence of the network is abstracted into a central unit which controls the network devices such as switches. SDN offers network flexibility as compared to the limitations of traditional networks.

In traditional networks, a router or switch as a single entity contains both the control plane (the brain) which takes the decisions as well as the data plane or the forwarding plane which is responsible for the forwarding of the data packets [7]. With the arrival of SDN, there is decoupling of control and the data plane. In other terms there is a separate dedicated central controller which controls the forwarding switches. The forwarding devices in data plane are dumb. They can be designed to have little or no intelligence. This idea has so many advantages. It offers network administrators the flexibility to program the networks to suite more specific or customized requirements. Also, the central unit has a global view of the network compared to legacy networks, where the switches only have a partial view of the network. This allows the controller to make decisions that effectively utilize network resources.

2.1.1. SDN Architecture

SDN Architecture has 3 major planes. They are the application plane, control plane and forwarding or data plane [8]. Each of these planes perform a unique function. The data/forwarding plane contains the switches where packet forwarding takes place. The control plan has the central hub unit termed controller. It lies in between the application layer and the data layer. This layer serves as the brain of the setup. It takes policies expressed by applications at the application layer and translates them into actions(rules) for the data plane to carry out. The central controller communicates with switches in the data plane via southbound API and with the Application plane via northbound APIs. The Application plane contains the network applications. These applications could run on physical or virtual hosts.

The controller here has the full knowledge of the network. It has a global view. Usually multiple controllers are employed each for a specific domain because of scalability issues. In large networks, more than one controller is used to control all the switches. This architecture is termed as Distributed controller network and the controllers communicate with one another using east-west API's [7]. The controller uses its global view to optimize packet flow through that network. The controller tells the switches in the data plane where

each packet should be forward by installing flow tables on them. This process is based on the OpenFlow protocol. The flow table has multiple fields consisting of the header field, action field, priority field, timer field and the counter field. SDN Architecture is showed in the figure below.

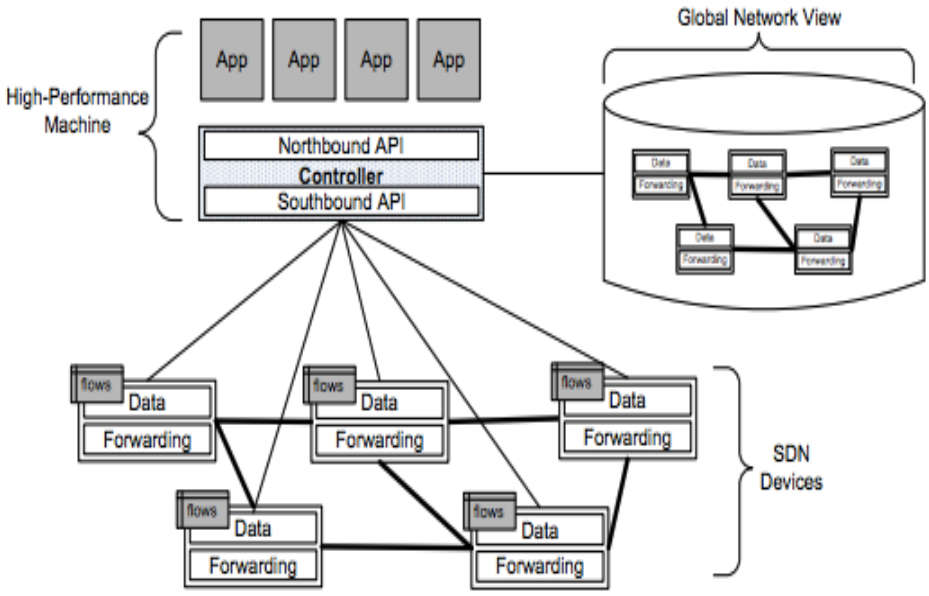

Figure 2: Architecture of SDN [9]

2.1.2. SDN Strength and challenges

Software defined networking gives the network more flexibility. Network administrators can implement policies quickly and easily. Mitigation of attacks also becomes easier. Software defined networks also provide a framework to incorporate artificial intelligence for traffic engineering and to deal with malware attacks and other security concerns. Following are some the advantages of SDN:

- Programming of user specific applications becomes easier since the abstraction can be shared which is provided by the control plane.

- Different applications can be integrated easily. Also, the traffic can be effectively divided in the network and hence load balancing along with traffic engineering becomes easy.
- The cost of the network hardware decreases as the switches and routers are not that expensive.
- The forwarding devices in the data plane can be easily integrated with the controller. No need to check for the compatibility of devices and hence integration becomes easy [7].

SDN also have certain challenges. One major challenge is the single point of failure. Since SDN uses the concept of centralization, the moment the controller, which is the brain of the network goes down, the whole network goes down. The issue of single point of failure is also a major security risk. Attackers can mainly focus on taking down the controller in order to bring the whole network down. Another challenge is scalability. As the network size increases, one controller will not be able to effectively handle the network responsibilities. More controllers are added as the network expands. Adding more controllers also raises the challenge of where to place the controllers. This is popularly termed as the controller placement problem. Several approaches to dealing with the controller placement problem have been presented in [10] and [11]. Security in SDN networks have also garnered a lot of research contributions in the past years.

2.2. Machine Learning

Arthur Samuel in 1959 described Machine Learning as: “the field of study that gives computers the ability to learn without being explicitly programmed.’’ This can be thought of as an older, informal decision. Tom Mitchell in his book in 1997 provided a more modern definition for Machine learning as:” A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.” Machine learning (ML) is a field of Artificial intelligence and can be defined as the scientific

study of algorithms and statistical models that computer systems use to perform a specific task without using explicit instructions, relying on patterns and inference instead [12].

Machine Learning is basically classified into four categories: supervised, unsupervised, semi-supervised and reinforcement learning. Brief explanations of these algorithms are outlined below.

Supervised Machine Learning: A supervised learning model is built by supplying the "system" with a "training data'' i.e. inputs and their known outcomes/outputs to enable it to create a relation. It is then provided new inputs to predict based on the relationship learnt. The training data is described as "labelled''. Supervised learning problems are categorized into "regression'' and "classification'' problems. In regression problems, the model tries to predict results within a continuous output whereas in classification, the model predicts results in a discrete output. Some of the commonly used supervised learning algorithms include decision trees, k-nearest neighbors, random forest, neural networks and support vector machines [13].

Unsupervised Machine Learning: Unsupervised machine learning is the exact opposite of the supervised approach. In this model, the system is given a set of input data without their corresponding outcomes. The model clusters the data based on relationships among the "features'' in the data [14]. Popular examples of such algorithms include k-means and self-organizing maps.

Semi-Supervised Machine Learning: Semi-supervised learning is midpoint between supervised and unsupervised learning. In this model, part of the input data has their corresponding outputs(labelled) whiles the remainder do not have their corresponding outputs(unlabeled). Example of such algorithms are 'transductive' SVM and graph-based methods.

Reinforcement learning: It is the task of learning how agents ought to take sequence of actions in an environment in order to maximize cumulative rewards [15]. An agent relates with an environment to learn the best actions to take to maximize the long-term reward.

Machine Learning has been applied in several fields to improve human living. Examples include the use of machine learning for natural language processing (NLP), medical diagnosis, customer segmentation, product recommendation and facial recognition. A typical machine learning application process entail:

- Data collection and preprocessing
- Feature engineering and training model choice
- Training and evaluating the model
- Fine tuning the trained model and
- Using the model to predict new instances

2.2.1. Data collection and preprocessing

Data collection and preparation is an integral part of building machine learning models. Based on the outlined machine learning problem, corresponding data is collected. A lot of datasets are available online depending on the research problem. Data is required to be cleaned and preprocessed to make sure the ML model is being feed with valid data that correctly models the problem considered.

2.2.2. Feature Selection and Engineering

Feature Selection and engineering is another vital step in Machine learning process. A feature is an individual measurable property or characteristic of a phenomenon observed [16]. It is information that can aid in the prediction or classification. Feature engineering is the process of using domain knowledge of the data to create features that make machine learning algorithms work[17]. The goal is to achieve features that improve the accuracy of the model. In feature engineering, better features can be created by combining other features in the dataset.

Feature selection aims at removing irrelevant and redundant attributes for the purpose of increasing learning accuracy [42] [43]. Feature selection enhances the learning accuracy and speed up the learning process of algorithm. Reduced number of features results in low

computational requirements, a feat that will reduce the burden on computing resources. There are three main groups of feature selection techniques. They are:

2.2.2.1. Filter Methods

These are feature ranking techniques that evaluate the relevance of features by looking at the intrinsic properties of data independent of the classification algorithm [42] [44.]. They use a ranking criterion to score the variables and a threshold is used to remove the features below the variable [42] [43]. Filter methods can be univariate or multivariate. Univariate filter methods ignore feature dependencies and hence can select redundant features whiles multivariate methods account for feature dependencies in their selection [42][44]. The filter methods are fast, scalable and independent of the learning algorithm but can provide lower accuracies since there is no interaction with the classifier [42][43]. Examples of filter methods are information gain, gain ratio, correlation-based feature selection, Markov blanket filter (MBF) etc.

2.2.2.2. Wrapper Methods

These methods use the predictor as a black box and its performance as the objective function to evaluate the variable subset [42][43]. A search procedure in the space of possible feature subset is defined and various subsets of features are generated and evaluated. The evaluation of a specific subset of features is obtained by training and testing a specific classification model [42][43]. This approach unlike filter methods considers the classification algorithm. A drawback is the computational cost involved. Some examples include sequential search algorithms and heuristic search algorithms.

2.2.2.3. Embedded Methods

Embedded methods consider the learning algorithm. It investigates the relationships between input features and the output as well as feature dependencies. It uses the independent criteria to decide the optimal subset for a known cardinality and uses the learning algorithm to determine the best optimal set among the optimal subsets across

different cardinality [42][43]. They are less computation intensive as compare to wrapper methods.

A hybrid approach is sometimes employed in feature selection where filter methods and wrapper methods are applied in a sequence to identify relevant feature [42][43]. These are termed as hybrid methods.

Filter methods are mostly used in cases where there is a large set of features. Our dataset will consist of 23 time-based features which is considered small so we will use wrapper and embedded methods for feature selection. A wrapper method based on forward selection and a reliable feature importance from tree-based models called SHAP proposed in [45] are used for our feature selection in our classification problem. These two approaches are chosen because of their popularity.

2.2.3. Choosing a Training Model

The next step is choosing the best algorithm to use. There are several algorithms to choose from. A ML algorithm can be based on supervised, unsupervised, semi-supervised or reinforcement learning. Figure 6 outlines a list of commonly used ML algorithms. The ones widely used in network traffic classification and routing are Random forest, Decision Tree Classifier, K-Nearest Neighbors and Neural networks.

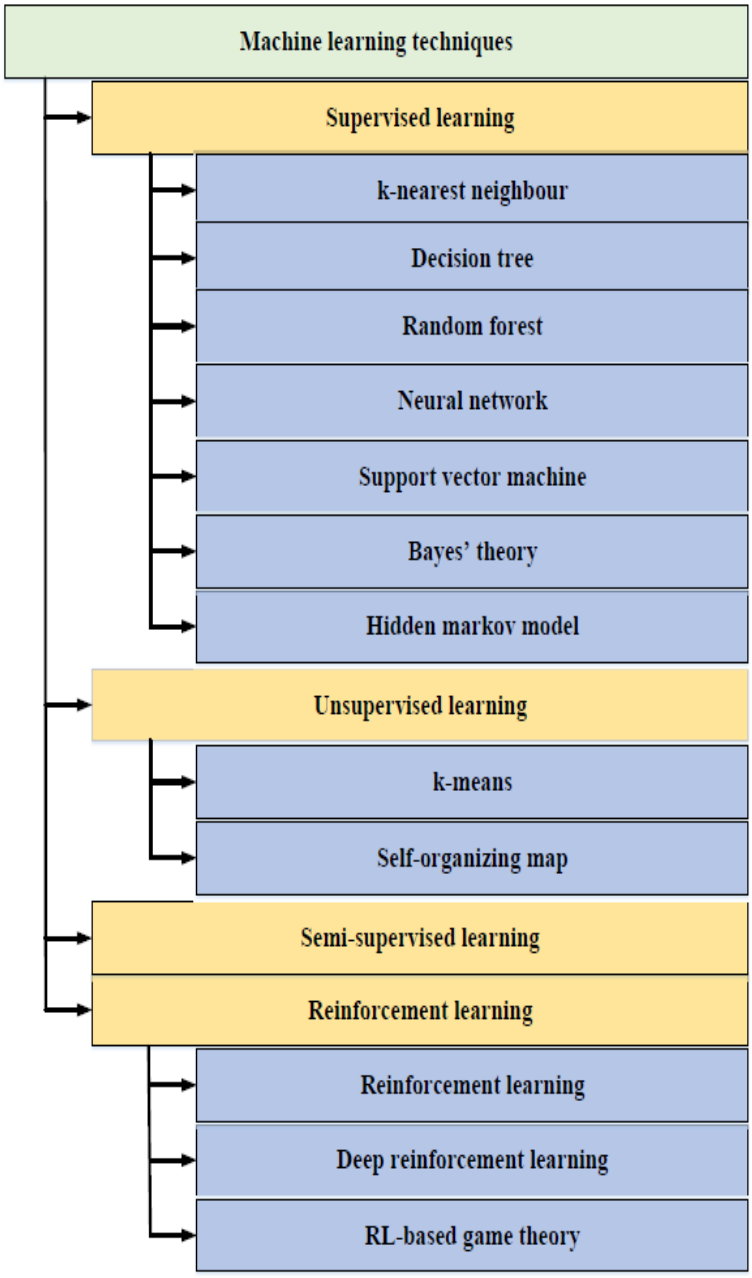

Figure 3: Common machine learning algorithms [5]

2.2.3.1. Decision Tree classifier

Decision trees are supervised learning algorithms making decision rules from fed data. Decision trees consist of leaves and decision nodes [54]. A leaf represents a label whiles the decision nodes and sub decision nodes represent conditions for possible outcomes nodes [54]. Decision Tree classifiers compare the features at each node and take new

branch until a leaf (label) is reached [54]. Using decision trees have some advantages which include:

- Decision trees are simple to understand and interpret [19].
- The cost of using the tree is very low (logarithmic in number of data points that are used for training the tree) [19].
- Able to handle multi-output problems.
- Requires less data preparation. Most of the techniques require creation of dummy variables, normalizing data etc. [19].
- Performs well even if its assumptions are somewhat violated by the true model from which the data were generated [19].

Disadvantages also include:

- Decision-tree learners can create over-complex trees that do not generalise the data well. This is called overfitting [19].
- Decision trees can be unstable because small variations in the data might result in a completely different tree being generated [19].
- The problem of learning an optimal decision tree is known to be NP-complete under several aspects of optimality and even for simple concepts. Consequently, practical decision-tree learning algorithms are based on heuristic algorithms such as the greedy algorithm where locally optimal decisions are made at each node. Such algorithms cannot guarantee to return the globally optimal decision tree [19].

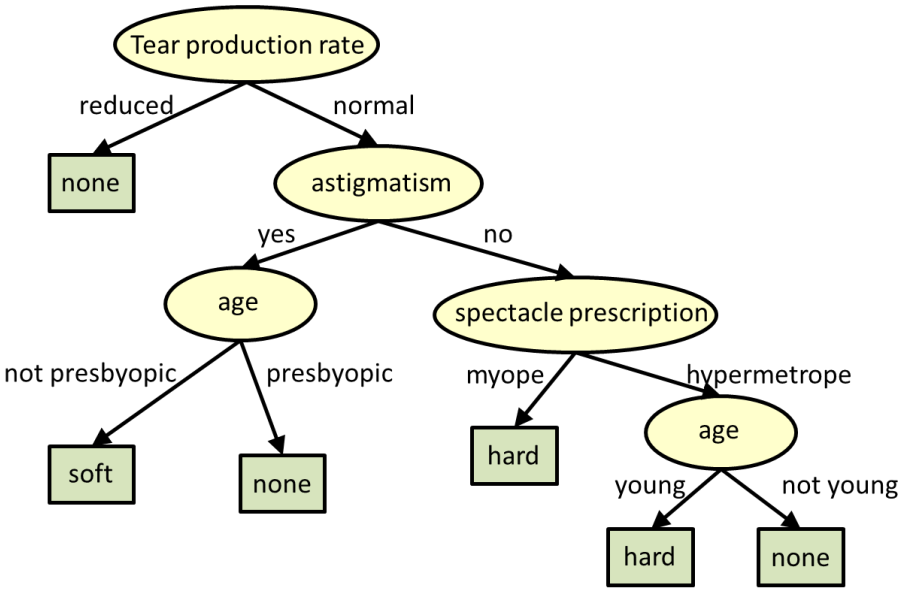

Figure 4: A sample decision tree

In conclusion, decision trees aim to create a model which predicts the value of target variable by learning simple decision rules that are inferred from the data features [19].

2.2.3.2. Random Forest Classifier

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting [55]. Random forest is an ensemble ML algorithm. It is one of the most powerful and commonly used machine learning classifiers. Random forests improve on the overfitting problem found with decision trees.

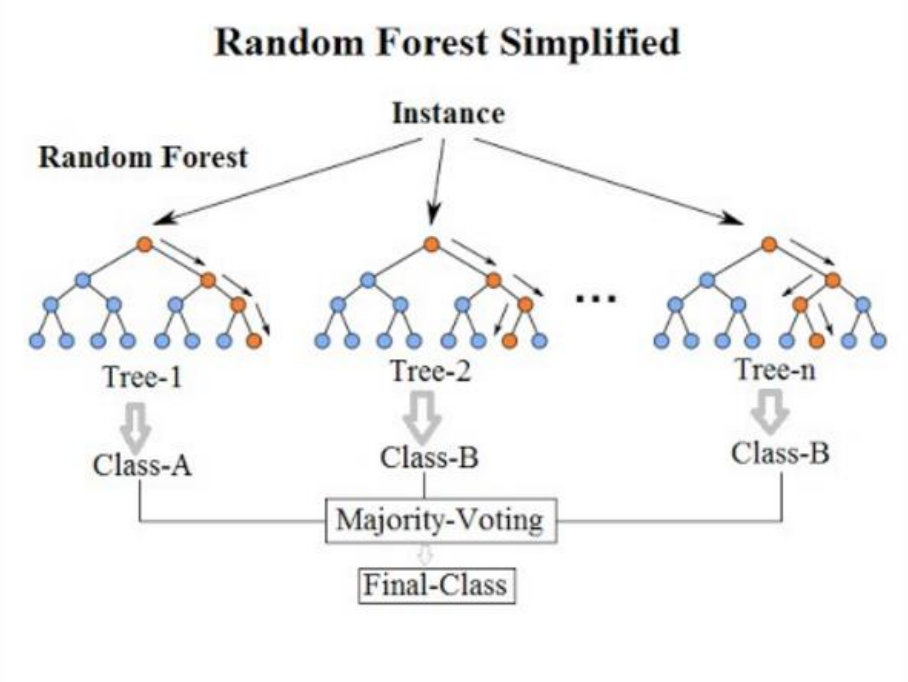

Figure 5: Random Forest

2.2.3.3. Artificial Neural Networks

Artificial neural networks (ANN) or connectionist systems are computing systems that are inspired by, but not identical to, biological neural networks that constitute animal brains. Such systems "learn" to perform tasks by considering examples, generally without being programmed with task-specific rules [53]. They mimic the biological concept of neurons to draw relations between a set of given data and an output.

Neural networks learn from input data. It learns by comparing the input and desired outputs and adjust its weights to fit them. Neural networks can learn via three major learning algorithms. These are *supervised*, *unsupervised*, and *hybrid* methods.

Supervised and unsupervised learning have already been presented in the preceding session. Hybrid learning is the third learning approach for Neural Networks. It uses both supervised and unsupervised learning. The network first learns in an unsupervised fashion to find good initialization points. This process is termed as pre-training. Supervised learning is then applied to fine tune the predetermined initialization points.

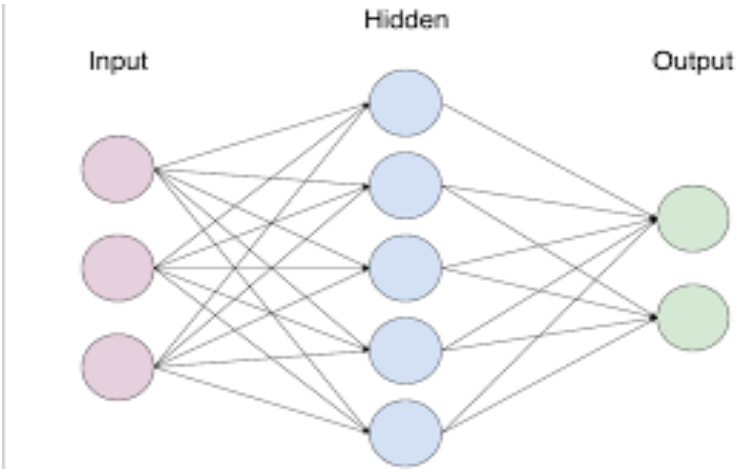

Figure 6: Artificial Neural Network

2.2.4. Performance Evaluation

After a model is trained, it must be tested to verify its performance. The most common metric used in traffic classification and routing is accuracy. It can be flow based accuracy or byte-based accuracy. Accuracy measures how correct or precise the model is. Error rate is another common metric used and can be determined by subtracting accuracy from 1. Other used metrics include precision, recall and F1 score. The choice of performance metric for a machine learning project is mainly dependent on the goal of the project.

Several of the above stated ML algorithms have been applied in network management over the years. Regarding our research, we will provide review on those works which applied machine learning techniques for traffic classification or route optimization. The review is organized as follows:

First, we will look at works that applied ML techniques to solely classify traffic. Secondly, we review those that utilized ML for just route optimization and finally we will look at works like our research that used ML to classify traffic and to determine the optimal route based on the traffic class.

2.3. Traffic Classification

Network traffic classification is a vital network activity that has potential of offering a foundation to deal with certain network management issues like QoS provision. With traffic classification, network administrators have knowledge of the network traffic profile and can provide swift solutions and allocate resources efficiently and effectively.

Over the years, the common IP traffic classification techniques used are, port-based method and Deep Packet Inspection (DPI) [20]. Port-based method classifies traffic or identify applications using TCP and UDP port numbers. However, many applications have started using varying or obscure port numbers which makes this method ineffective. DPI entails an investigation of the packet's content and comparing it to certain known patterns. Though DPI has high classification accuracy, it involves high computation and uses a lot of system resources. Also, most traffic is encrypted lately due to privacy concerns and hence the packet's content cannot be assessed making DPI ineffective.

A more recent method proposed by researchers is classification based on Machine Learning algorithms. ML-based approaches use the statistical properties learnt from data to identify traffic classes. It does not require the packet content and hence can classify encrypted data. Machine learning leverages the "OpenFlow'' protocol in SDN, to collect data for analysis and can then predict traffic class at a low computation cost. A lot of research has been done to classify traffic. We have presented a few below grouped based on the ML category applied.

2.3.1. Supervised Learning based traffic classification

Supervised learning-based traffic classification aims to identity traffic classes using data that has known outputs. The dataset used in such classifications have their targets mapped to the input data. Majority of ML based traffic classification works are based on supervised learning.

[21] presents a traffic classification work based on Bayesian analysis techniques. Their paper used basic and refined forms of Naïve Bayes algorithm to classify network traffic

into different categories based on their applications. The data used was experimentally acquired using a network monitor on a site referred to as ‘Genome Campus’. The campus consisted of 1000 users connected to the internet via a full-duplex gigabit Ethernet link. The captured traffic content was reviewed to obtain the target data (corresponding application) of the flows. Basic form Naïve Bayes, Naïve Bayes with kernel density estimation and Naïve Bayes after FCBF filtering were the 3 classification approaches applied to the data. The evaluation metrics used were flow-based accuracy, byte-based accuracy and trust. Their approach yielded a flow-based accuracy of 96.29% on the initial dataset and an 93.78% accuracy when it was applied on dataset captured later demonstrating the “temporal stability’’ of their method. Even though [21] yields a very good accuracy, their classification was based on applications and this becomes impractical as the number of applications on the internet increases exponentially.

Authors in [22] present a classification based on whether the flow is an elephant one or not in an SDN system. They used a cost-sensitive learning technique to improve the accuracy in detection. The classification is done in two stages. The first stage uses head packet measurement to distinguish between elephant and mice flows. Flow statistical features are extracted from suspected elephant flows. Optimal features are selected using the correlation-based filter. Cost sensitive decision trees are then applied to confirm if the suspicious flows are truly elephant flows or not. Their work achieved around 95% accuracy This approach is more applicable in data centers where quick flow scheduling is priority.

A supervised learning-based approach is presented in [23], [24] and [25] to identify the applications of traffic flows in software defined networks (SDN). [23] presents an SDN architecture that allows traffic data collection based on OpenFlow. Multiple supervised learning algorithms were then on the collected data to classify the flows into different applications. The evaluation was done in an enterprise network. [24] uses a 2-stage approach using a combination of ML classifier and a DPI classifier. The paper leverages the speed of ML and the accuracy of DPI for more effective and efficient classification. When a flow arrives, the ML classifier does the classification. The ML classifier returns

the result as well as a reliability value. If the returned reliability value is greater that a set threshold value, the system accepts the ML classifier results as the application of the flow. If the returned reliability value is lesser than the set threshold, the flow is passed to the DPI for classification. If the DPI returns a class, that is assigned to the flow and if it returns "UNKNOW', the ML result from the first stage is given to the flow. They applied the multi-classifier on multiple datasets and had a consistent accuracy of more than 85%. This approach is cannot be used when the traffic content is encrypted since DPI requires access to the traffic content. Also, DPI is computation intensive and hence will be difficult to use in real-time environment. In [25], SVM algorithm is used to classify UDP traffic. The UDP traffic is based on cisco NetFlow records. The classifier simply uses packet counts and bytes exchanged with other hosts to distinguish the behavior of an application. The classification accuracy was over 90%. Their work focused solely on UDP traffic.

The authors in [26] present another application-based classification. They outlined a framework called Atlas. Atlas was used to identify mobile applications. They used crowd sourcing to collect ground truth data from mobile users. They achieved that by installing mobile agents on end user devices to collect 'netstat logs' belonging to each application running on the device. OpenFlow collected the flow features whiles running on the wireless AP. Both the flow features and 'netstat logs' were sent to the control plane of the SDN, to compose the training data for the ML application. Decision tree algorithm was applied on the training data to build a classifier. The classifier was then used to identify the mobile applications of the traffic flows. The classifier was evaluated on the 40 most popular applications on the google store and it achieved an accuracy of 94%.

The number of applications on the internet keeps growing every day. It is therefore not feasible to build classifiers to identify all applications. Most of the application-aware works are built to classify a list of common applications and will not be of much use in an SDN-IoT networks where a wide range of applications exist.

2.3.2. Unsupervised Learning based traffic classification

Unsupervised learning-based traffic classification aims to identity traffic classes buy grouping data into clusters based on similar features. The dataset used in such classifications do not have the corresponding target information or ground truth.

[27] present a classification that uses cluster analysis to classify the traffic data into groups. They extracted data from two distinct packet traces, a publicly available trace called Auckland IV and full packet trace they collected themselves from the University of Calgary. They extracted TCP-based application flows from the packet traces. The statistical features of the flow used included total number of packets, mean packet size, mean payload size (excluding headers) etc. They applied K-means, DBSCAN and Autoclass algorithms on both datasets and calculated the overall accuracy (defined as the sum of True positive for all clusters divided by total number of connections). Connections that did not fall within any cluster was labelled as noise. After clustering, the traffic class that occurs majorly in a cluster is used to label that cluster.

Another unsupervised based classification can be found in the work presented in [28]. Just as presented in [27], they extracted flows from a packet trace and acquired a range of statistical attributes like packet size statistics, interarrival statistics, byte counts, connection duration etc. The attributes are then used by the EM clustering algorithm to group the flows into clusters. They went further to refine the clustering by generating classification rules that group the clusters based of raw data. Based on the rules, they identify input attributes that do not have a significant impact on the clustering and remove them. The process is repeated to fine-tune the clustering.

In [52], an unsupervised profiling approach is used to investigate the flow features and link patterns in a short time window. They formulated the traffic classification problem as a graph co-clustering problem with topology and edge attributes and then used a hybrid flow clustering (HFC) model.

The challenge with unsupervised learning is that the user must spend extra time to interpret the results. The user must spend time interpreting and label the classes following the classification. This can result in misclassification errors and labelling relevant flows as noise

2.3.3. Semi-supervised Learning based traffic classification

Semi-supervised classification uses data that has a part of it mapped to targets and the remainder unmapped.

[29] presented a framework for a QoS-aware classifier based on semi-supervised machine learning. The aim of the work was to identify the QoS classes of traffic flows. Their worked assumed that applications that require the same QoS have similar statistical attributes. First, the edge routers identified whether a flow was an elephant one or not based on the percentage of bandwidth the flow uses. The elephant flows were identified, and their flow information were forwarded to the SDN controller via OpenFlow protocol. Statistical features were then extracted from the flow information. DPI was applied to some of the packet traces to identify QoS classes and labelled. A semi-supervised ML algorithm, Laplacian SVM was applied on both the labelled and unlabeled data to build the classifier. Simulation results showed that the classifier had an accuracy of over 90%. Their work did not account for all classes of QoS since only elephant flows were considered.

A similar work is presented in [30]. The significant difference between their work and [29] is the learning mechanism used. They used a refined form of tri-training learning mechanism called heteroid tri-training. Tri-training is a classic semi-supervised learning mechanism that uses partially labelled datasets and leverages three identical classifiers to enable iterative training [30]. In their approach, they used three different classifiers instead of three identical classifiers. The classifiers used were SVM, Bayes classifier and K-Nearest Neighbor. Their approach gained around 86% accuracy on a 20% application unknown dataset, around 92% on 40% application unknown dataset and a 72% on a 60%

application unknown dataset. The accuracy of the model decreased as the number of unlabeled data increased.

In [39], the traffic classification task is modelled as a multitask learning problem. Values such as bandwidth requirement and duration of flow which are easy to measure are predicted alongside the classification class. They used a used large amount of labelled bandwidth and flow length datasets to large volume of unlabeled traffic class dataset. This approach was to leverage the cost involved in labelling data for supervised learning.

Semi-supervised learning-based approaches require that certain assumptions be made about the data. This assumption can be abused resulting in the model performing poorly on unseen data.

2.4. Routing Optimization

Routing is a vital network function. In legacy networks, routing decisions are made by switches in decentralized manner. In SDN, the central controller makes routing policies by installing flow tables on the switches. Optimizing routing allows for effective use of network resources and QoS provision. The SDN architecture takes advantage of the controller's global view of the network to make efficient routing decisions.

Heuristic algorithms like ant colony optimization are one of the most effective routing optimization approaches [33]. Heuristic solutions give close to optimal solution, but they involve a high computational time cost and hence unable to calculate optimal solutions in real time [33]. This drawback of heuristic algorithms makes impractical to run such algorithms in the SDN controller which is responsible for calculating the route for flows. Machine learning operates on building relationships between features and does not need the exact network model as heuristic algorithms. Also, once a ML model has been trained, it is able to predict near-optimal solutions in a very short time making it effective for real time use. Several studies have been conducted on using ML especially Reinforcement learning to solve the routing optimization problem. Related studies have been presented below grouped based on the ML approach used.

2.4.1. Supervised learning-based Routing optimization

Supervised learning-based routing uses labelled datasets. The data usually consists of network and traffic conditions as inputs and their corresponding outputs obtained by using heuristic routing algorithms. This helps model a ML solution that gives heuristic-like routing solutions.

[32] presents a routing framework called Neuroute. The Neuroute used a traffic matrix predictor based on a Long Short-Term Memory Recurrent Neural Network (LTSM-RNN). The model took in traffic matrix at time t and output the traffic for time t+1.LTSM-RNN was trained on sampled training data using backpropagation to build the predictor model. The predicted network traffic and networks states as inputs and their corresponding heuristic routing solutions as outputs are used to train a Deep Feed Forward Neural network to build a traffic routing unit. The architecture was implemented on the GEANT network topology. The work focused on effective routing of traffic volumes and necessarily meeting QoS requirements of specific traffic classes.

[33] presents another optimized routing solution based on supervised learning. They presented a framework with ML based meta-layer for routing in real-time. The architecture the output of a heuristic algorithm and its corresponding network information as input data to build a neural network that can provide heuristic-like routing decisions. Their proposal was applied on the NSTNET and compared to some other routing solutions and the running time was found to be faster.

2.4.2. Reinforcement learning based routing optimization

In RL, an agent in an environment makes decisions (actions) and is rewarded based on the action taken. The goal of the agent is to make decisions that will result in a high positive reward. RL is mostly applied in solving optimization problems. In the routing optimization in an SDN, the SDN controller is the agent and the network are the environment. The agent takes information (state space) from the environment which in this scenario consists of

network and traffic matrices. The agent makes a routing decision (action) and the reward may be the network delay or any outlined network evaluation parameter. Some works that utilize reinforcement learning for routing optimization have been outlined below.

[34] presented a reinforcement learning based distributed routing based on SDN. Though in an SDN topology, they implemented the routing protocol to run in a distributed manner. The RL algorithm was added to the OSPF routing protocol and tested. They compared it with the traditional OSPF routing protocol and their approach achieving better QoS delays and jitter.

[35] presented a routing approach based on RL learning algorithm. The optimization was implemented in a multi-layer hierarchical SDN architecture. The architecture choice was to reduce signaling delay. The SDN controllers were designed in three levels; the super, domain/master and slave controllers as shown in figure below. Each of these controllers had their unique set of responsibilities to reduce the burden on the super controller. The slave controllers provided read only access to the data plane and received port messages from them. The slave controllers also performed simple control functions as traffic admission, flow control etc. The domain controllers received flow set-up requests and installed flow rules on switches. The super controller had a global view of the network and controlled full network functions. A QAR algorithm with QoS based reward was implemented in the super controller to achieve optimal routing solutions. The proposal was deployed in a real Sprint GIP network and tested. The approached provided fast convergence as compared to existing learning solutions.

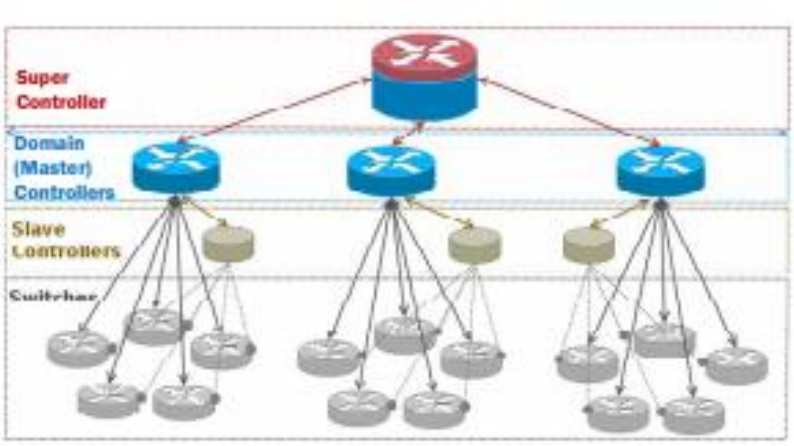

Figure 7: Distributed hierarchical architecture [35]

[36] also presented an approach based on deep reinforcement learning. The paper used a DRL mechanism called DDPG to determine optimal routing solutions. The optimization mechanism outlined in the paper is called DROM and experimental results proved that it had good convergence and effectiveness

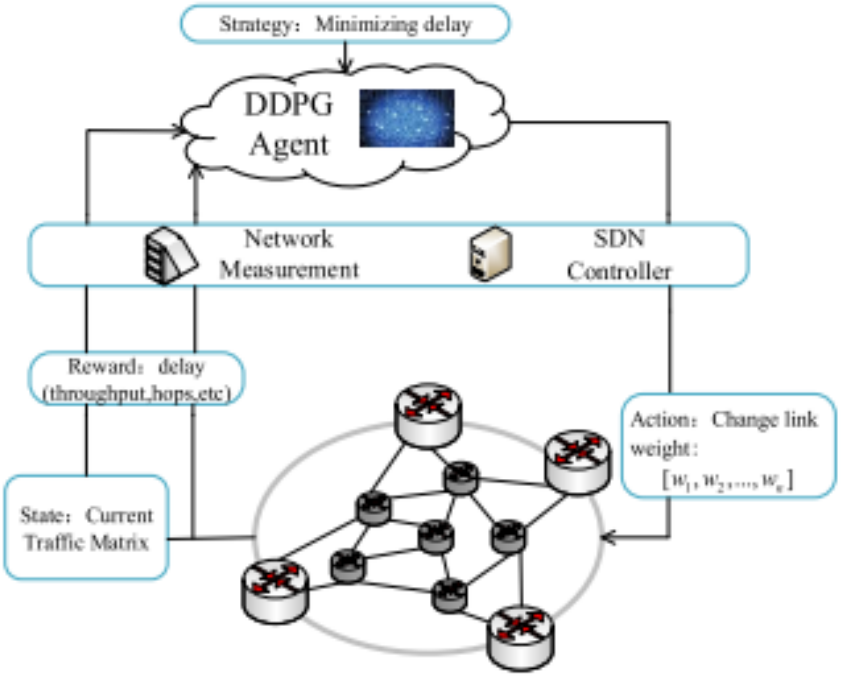

Figure 8: DROM framework [36]

2.5. Traffic Classification Based Optimized Routing

In this section, we look at works that used ML to classify traffic and went on to determine the optimal route based on the traffic class determined.

[37] utilizes ML and SDN to improve scheduling and control in data centers. The framework involved two steps. The first step involved using Machine learning at the edge of the network to classify the traffic. The classification result is the used by the centralized SDN controller in conjunction with its global view of the network to implement optimized routing solutions. This paper did not provide specifics as to which algorithms and mechanisms will be used for the classification and routing.

Another work that combines traffic classification and route optimization is presented in [38]. They proposed a network traffic classification based on a Deep Neural network (DNN) classifier in an SDN environment. A VNF is introduced in the architecture to reduce the burden on the SDN. When a flow arrives, the SDN switch routes the flow based on the flow rules installed. If there is no flow rule for the packet, it sends a request to the controller. The controller calculates the path to route the flow based on the network topology and installs these flows in the SDN switches. The controller also installs a flow rule to pass a copy of the packet to a port where the VNF will be listening. The VNF captures the traffic and extract features for traffic classification based on the trained DNN. The identified class information is tagged in the DSCP field and sent to the SDN controller. The SDN controller now searches for a path that will meet the QoS requirements of the identified traffic class and installs those flows with high priority in all switches along the path giving them priority over the initially installed flows. This work presents an approach to improve the network quality, but the work does not propose any optimization approach that as used by the SDN controller after classification to determine the path. The controller simply uses its default routing mechanism based on the global view to route the identified packet.

2.6. Conclusion

In this chapter, we gave an overview of SDN architecture and compared it to traditional networks. We outlined its strength and challenges. We presented a review on Machine learning and the various steps involved in designing a Machine learning application. We have also given reviews on research works done so far in the area of traffic classification and route optimization in SDN networks. In the next chapter, we present our proposed approach for classification based optimized routing in SDN-IoT environments.

3.0. QoS Aware Traffic Classification

In this chapter, we describe the framework for our intelligent traffic classification based optimized routing. Our framework looks to classify traffic flow into QoS classes and determine the routing path based on the flow QoS requirements. In an SDN-IoT, the central controller is the brain and decision maker of the set up. It controls all the functions of the data plane. In our architecture, there will be 2 major decision-making process; traffic flow classification and optimized routing. To avoid burdening the controller, we adopt a VNF as used in [38] to handle the traffic classification and then the controller will handle the route optimization problem. Virtual network functions (VNFs) are virtualized tasks formerly carried out by proprietary, dedicated hardware. VNFs move individual network functions out of dedicated hardware devices into software that runs on commodity hardware [31]. The VNF will run on a server deployed in the data plane. As already outlined, our model will perform:

1. Efficient QoS-based traffic classification, and
2. Optimized routing.

In this chapter, we present a traffic classification based on QoS requirements of the traffic. The goal is to apply ML algorithms on the chosen dataset for a QoS based classification. The effectiveness of multiple supervised learning algorithms on the network traffic data is tested. The feature choice as well as the how the number of features affect the classification are discussed. Finally, we discuss the amount of time required for training and classification by the various ML algorithms and how feature selection impacts the time.

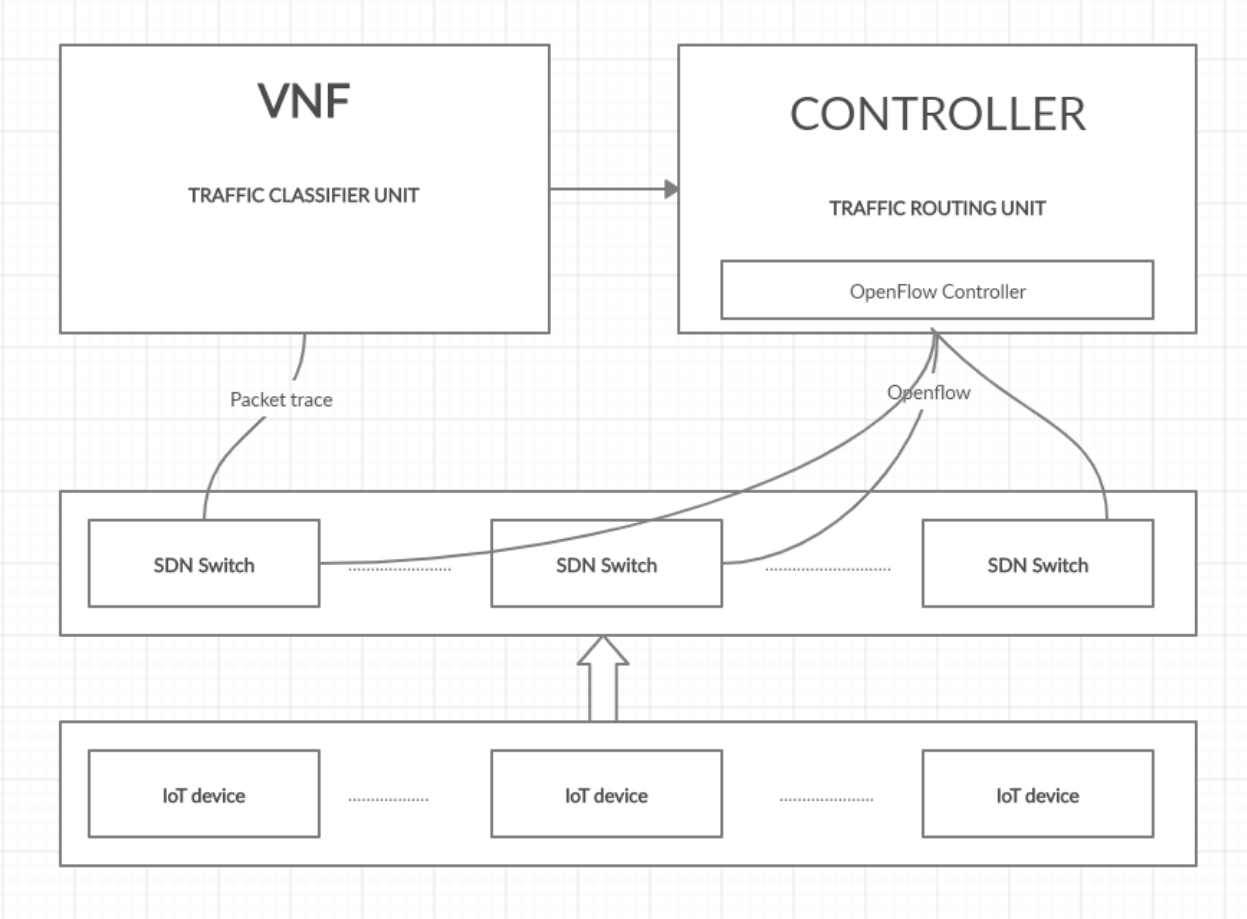

Figure 9: System Design

3.1. Classification Process

In a real time, the classification process consists of the data collection phase, classification phase and algorithm training phase.

3.1.1. Data collection phase

In our architecture, the SDN controller installs a low-priority flow rule on the edge SDN switches in the network. The low priority flow rule is a path from the source port to a specific port that is being monitored by the VNF server. When a flow arrives at the SDN switch, the switch inspects the packet and checks for a matching flow rule. If there is a flow rule with higher priority than the installed low priority rule to the VNF, it will route the packets through that path. If there is no higher priority flow rule, it will send a

PACKET_In message to the SDN controller and simultaneously send a copy of the packet to the port being monitored by the VNF.

The VNF will capture the packets for a timeout period and calculate the time features from the packet. The features are based on simple mathematically formulas and hence are simply calculated and fast.

3.1.2. Classification Phase

The pre-trained ML learning algorithm is applied on the extracted features to predict the QoS requirements of the traffic flow. In our work the classification is done according to the bandwidth and latency requirements of the flow. The VNF now sends the classification results of the flow to the SDN controller through a secure and dedicated channel.

3.1.3. ML Algorithm Training Phase

SDN-IoT traffic is highly dynamic and hence the need to constantly train the ML algorithm for effective classification. The training can be done online or offline and used online. For online training case, the VNF server can be armed with a database and a DPI tool. As packets come in, the server will store copies of the packets as training samples and use DPI to identify the ground truth. It can then use the traffic and ground truth set to train the ML algorithm during off peak periods. For offline training, the stored traffic will later be identified and used to build a classifier offline and uploaded unto the server later. This work does not fully investigate the two approaches and hence does not choose any of the two.

3.2. Approach

The strategic goal of this work is to present a machine learning approach that can classify traffic classes without having access to the network content. Owing to sensitive nature of IoT devices, security and privacy are major concerns as IoT device usage increases exponentially. We aim to identify key statistical features that can be used to identify traffic QoS class and test the effectiveness on ML algorithms of ‘covered’ traffic data like TOR

datasets. This experiment uses real world data to test the effectiveness of ML algorithms classifying encrypted traffic based on their QoS.

3.3. Dataset

We use the dataset provided by [40] which was captured in a real-world ToR network. Due to the sensitive nature of IoT traffic, we believe most IoT traffic data will be encrypted in the future for security purposes. We therefore chose an encrypted dataset. We also picked the ToR dataset because it is a commonly used encrypted dataset hence providing us works to compare our results to. The dataset contains 8 types of traffic (browsing, chat, audio-streaming, video-streaming, mail, VOIP, P2P and file transfer). The dataset is diversified and represents a comprehensive range of QoS classes. The dataset consists of more than 18 applications including Facebook skype, Spotify, Gmail etc. They used the architecture below to generate the dataset.

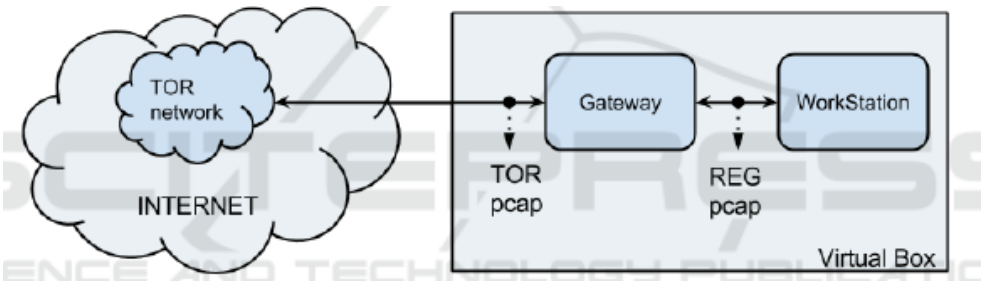

Figure 10: Tor capture scenario [40]

The data captured (pcap) look the same i.e. same source and destination IP address, source and destination port and same protocol (TCP). Since the data capturing was done in a controlled environment, the applications were executed one after another, so the traffic was labelled. However, the datasets include some noise i.e. wrong labelling and will affect our classification accuracy though not significantly. ISCXFlowMeter [41] was used to generate the flows and calculated all necessary parameters. Statistical time related features were calculated separately in forward and reverse directions. The data was presented for

different flow timeouts. Our focus will be on the flows with short flow durations based on the conclusion made by [40] that flows with shorter timeouts produced the best results in the classification problem.

Table 1: Contents of TOR datasets

	Browsing	Mail	Chat	Audio	Video	FT	VoIP	P2P	Total
5s	2645	497	485	1026	1529	1663	1529	2139	14508
10s	1604	282	323	721	874	864	2291	1085	8044
15s	264	213	273	50	681	556	1746	81	3864
30s	694	111	153	332	364	311	790	375	3130
60s	411	60	90	190	196	165	413	198	1723
120s	239	34	151	119	105	86	225	110	969

Each of the datasets consisted of 28 time-based attributes and a label. Time-based features have high speed of calculation when compared to some other features that can be extracted from flows. Time-based features require simple arithmetic to calculate them and can be derived from first few packets of a flow making efficient for near real time use. The features included:

- The inter-arrival times (IAT) in the backward and forward direction (mean, min, max, std), the time is the time between two packets sent forward or backward.
- Flow IAT which is the time between two packets in either direction (mean, min, max, std).
- Idle time which is the time a flow was idle before becoming active (mean, min, max, std).
- Active time, the time a flow was active before becoming idle (mean, max, min, std).
- Flow bytes/s, flow packets/s and flow duration.

3.4. Data Preprocessing

As learnt from our literature review, classification based on application becomes impractical as internet applications keep increasing. Classification based on QoS classes is more practical since different applications may belong to the same QoS class which has identical requirements. Classification based on QoS also provides a working model for even new applications since applications of identical QoS class are assumed to have similar statistics.

The dataset used has a default labelling based on the application. For the purpose of our work, we relabeled the data into a 2-output target based on QoS parameters; bandwidth and latency. The classification criteria are as follows:

Table 2: QoS classification

Application	Bandwidth requirement	Latency requirement
VOIP/chat	LOW	LOW
Mail/browsing	MID	MID
Audio/video	HIGH	MID
P2P/File transfer	HIGH	BEST EFFORT

The multi-output approach helps the ML algorithm to determine a relationship between the features used and each QoS requirement.

3.5. Feature Selection

Our dataset consists of 28 features. Since ToR datasets have the same source and destination IPs, same source and destination ports and same protocols, we will be dropping these features leaving us with 23 time-based features. The instances of our datasets vary as outlined in Table 1 based on the flow timeouts. All the remaining 23 time-based features

do not necessarily provide relevant information about the QoS class of traffic. It is extremely important to select the subset of features that have the most impact of the classification task. This process is termed as feature selection. In some context, our feature size is not that large to warrant feature selection but one of the goals of this session of our work is to investigate how different feature combinations contribute to the classification task. Also using feature selection will result to a reduced number of needed features hence reducing computation time which is very vital for real time use.

As already outlined in the previous chapter, a wrapper method based on forward selection and a reliable feature importance from tree-based models called SHAP proposed in [45] are used for our feature selection.

3.6. Classifier Algorithms

Three ML algorithms are applied in this research. These algorithms have been described in Chapter 2 and were chosen due to previous reported results as seen from our related works review. They include:

1. Random Forest Classifier
2. Decision Trees Classifier
3. K-Nearest Neighbors Classifier

Each of these algorithms will be applied on the dataset and the results compared for different feature subsets based on our feature selection techniques. The algorithm exhibiting the best performance will be chosen and tested on new data.

All the algorithms will be executed using their default parameters. Though tuning algorithms to achieve optimal solution is commonly practiced, we believe the default parameters produce acceptable results for the scope of this research.

3.7. Performance Metrics

As stated in Chapter 2, the commonly used performance metrics for Traffic classification are accuracy and F1 score. Accuracy is calculated as

$$\text{Accuracy} = \frac{\text{Number of correct classifications}}{\text{Number of All tested samples}}$$

For a multi-output classification task like ours, the considered accuracy is weighted correct classifications across both outcomes.

F1 score measures the test's accuracy. It combines both precision and recall scores of the test [46].Precision is the fraction of relevant instances among the retrieve instances whiles recall is the fraction of relevant instances that have been retrieved over the total amount of relevant instances [47]. We used the weighted F1 score to account for label imbalance.

$$\text{Precision} = \frac{\text{True positive}}{\text{True positive} + \text{False positive}}$$

$$\text{Recall} = \frac{\text{True positive}}{\text{True positive} + \text{False negative}}$$

$$\text{F1} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

The training time is also recorded. The training time is the total time taken to train an algorithm on a fold of the dataset.

3.8. Experimental Design

Scikit-learn, a machine learning library for python is used to run the experiments. The software environment is Jupyter Notebooks. All the experiments are run on a Dell OptiPlex 7020 with 8gb RAM, Intel R core i5 3.5GHz processor. The OS is windows 10. The system type affects the time used to train and test the algorithms. The dataset is split into training

and testing in 8 to 2 ratios. We use a 10-fold cross validation for training and testing our models. This approach is a common practice in ML projects.

3.9. Results and Analysis

This section discusses the results achieved from the experiment outlined.

The first part analyses the results from the experiment with all three ML algorithms applied to the training set without feature selection. The second part reviews the results obtained from the different feature selection methods. We then review the selected and choose an optimal set of features. Finally, we analyze the results obtained by applying our ML algorithms on the data with feature selection.

3.9.1. Algorithm Comparison without Feature Selection

The experiment is run for flow time-outs of 5s, 10s and 15s. Each of the data consists of 23 features. The accuracy, F1 score and algorithm training time are measured and shown. Figure 11 shows F1 scores and classification accuracies for the 3 ML algorithms for the dataset with 5sec timeout. The Random forest classifier achieves the highest F1 score of 0.914. The decision tree classifier has the best accuracy of 0.849. The least performing algorithm among the 3 is the K-nearest neighbor which records lowest values for both accuracy and F1 score.

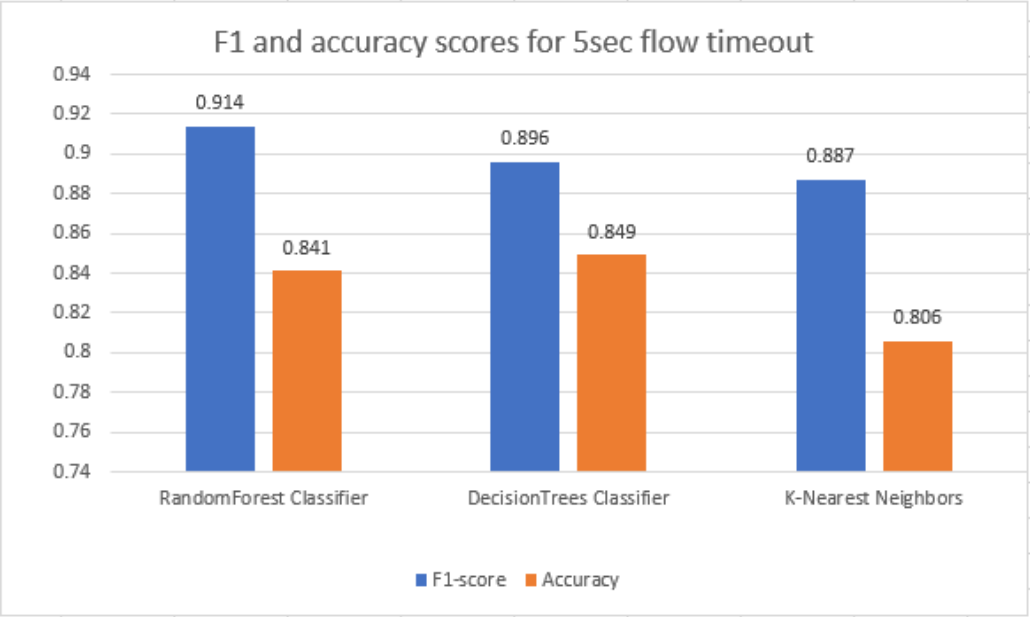

Figure 11: F1 and accuracy scores for 5sec flow timeout

Figure 12 shows F1 scores and classification accuracies for the 3 ML algorithms for the dataset with 10sec timeout. The Random forest classifier achieves the highest F1 score of 0.904. The decision tree classifier has the best accuracy of 0.838. The least performing algorithm among the 3 is the K-nearest neighbor which records least values for both accuracy (0.786) and F1 score (0.875).

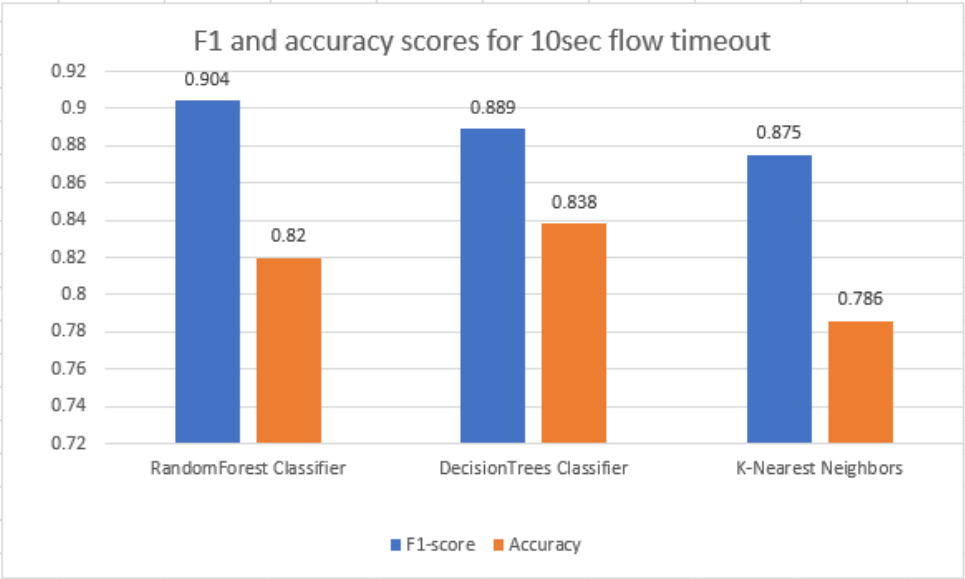

Figure 12: F1 and accuracy scores for 10sec flow timeout

Figure 13 below shows F1 scores and classification accuracies for the 3 ML algorithms for the dataset with 15sec timeout. The Random forest classifier achieves the highest F1 score of 0.912. The decision tree classifier has the best accuracy of 0.872. The least performing algorithm is the K-nearest neighbor which records least values for both accuracy (0.795) and F1 score (0.868).

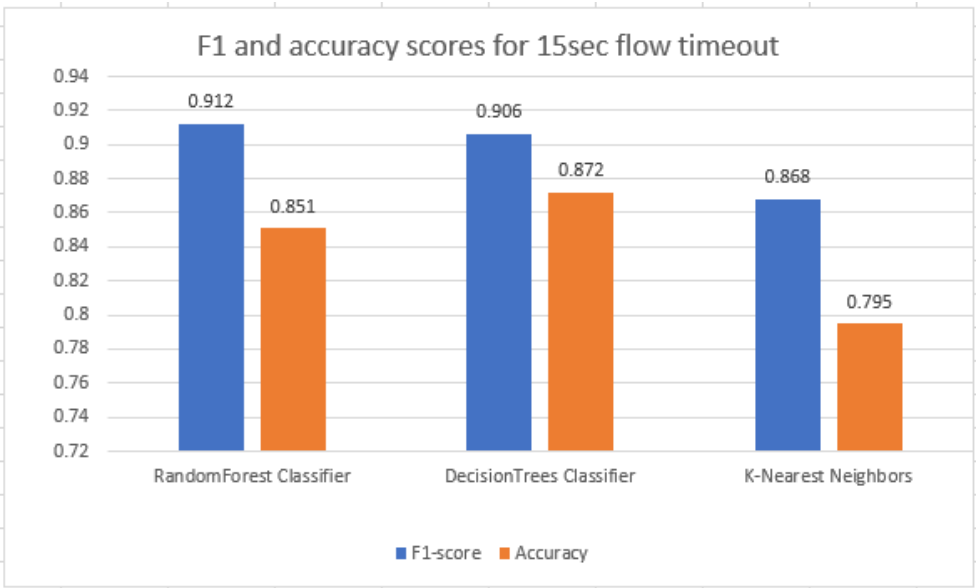

Figure 13: F1 and accuracy scores for 15sec flow timeout

From Figures 11-13, it can be observed that the Decision Tree classifier achieves the best accuracy in all instances whiles Random Forest classifier achieves the highest F1 score. The KNN performs poorly in all 3 timeouts in comparison to the other classifiers.

The time taken to train the algorithms for the different flow timeouts have also been recorded. Since intend to design a solution applicable in real time use, the training time is a key performance metric.

Table 3 shows the training times for the 5-sec timeout dataset. The fastest training algorithm is K-Nearest Neighbors and the slowest is Random forest classifier.

Table 3: Training times for 5-sec timeout dataset

Algorithm	Training time(s)
Random Forest classifier	0.30682
Decision Trees classifier	0.23785
K-Nearest Neighbors	0.02999

Table 4 and Table 5 also show similar results as seen in Table 3. The times for the various time outs are mainly dependent on the size of the training data. The 5 sec timeout dataset has more instances hence the training times are seen to be higher as compared to the 15-sec timeout dataset which has the lowest number of training instances.

Table 4: Training times for 10-sec timeout dataset

Algorithm	Training time(s)
Random Forest classifier	0.15292
Decision Trees classifier	0.10064
K-Nearest Neighbors	0.01099

Table 5: Training times for 15-sec timeout dataset

Algorithm	Training time(s)
Random Forest classifier	0.07198
Decision Trees classifier	0.04398
K-Nearest Neighbors	0.00698

It can be concluded from the 3 tables that, given a dataset, KNN trains faster than both Random forest classifier and Decision tree classifier. Random forest classifier takes the longest time to train. On the F1 and accuracy graphs, Random Forest and Decision tree classifiers achieve the best results.

3.9.2. Algorithm Comparison with Feature Selection

In this section, we look at applying feature selection to improve our algorithms.

As stated, we used a wrapper method based on forward selection and a reliable game-theory based approach called SHAP. Wrapper methods consider the ML algorithm been used. We have applied the wrapper method on the datasets for the different timeout values. We consider feature selection on the 2 algorithms that yielded the best results without feature selection i.e. Random forest classifier and Decision tree classifier.

3.9.2.1. Wrapper Method with Random Forest Classifier

With the Random forest classifier, we applied the wrapper method on the different datasets. The wrapper method was implemented using the SFS module in python. The sequential forward selection (SFS) starts with an empty set of features. It then increases the number of features in one increments and finds the performance of the different combinations of the features. For each number of features, it outputs the best set of features that gives the highest performance score. On the 5sec timeout dataset as shown in Figure 14, the best performance occurs at a combination of 6 features.

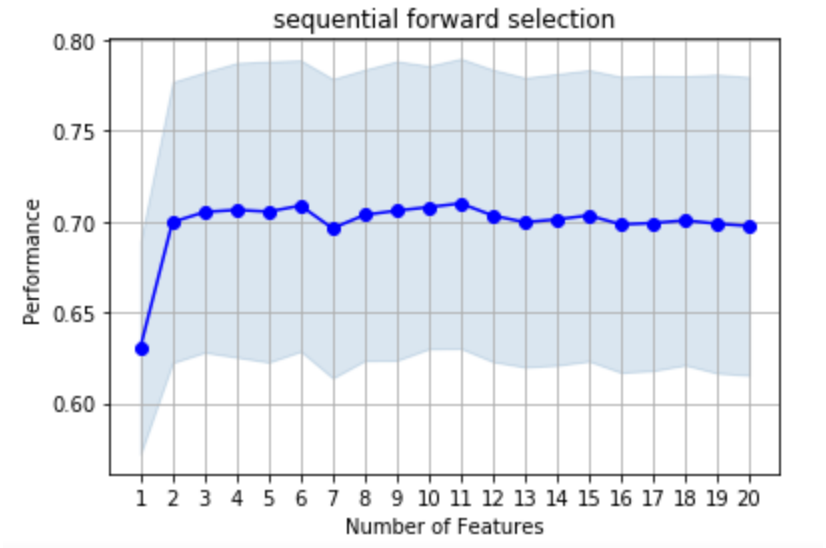

Figure 14: Wrapper method with Random forest classifier for 5s timeout dataset

In Figure 15, the peak performance occurs when using 16 features. The goal of feature selection however is to achieve excellent results with a considerable small number of features. The lowest number of features that gives a close to peak performance on our graph is 9 features.

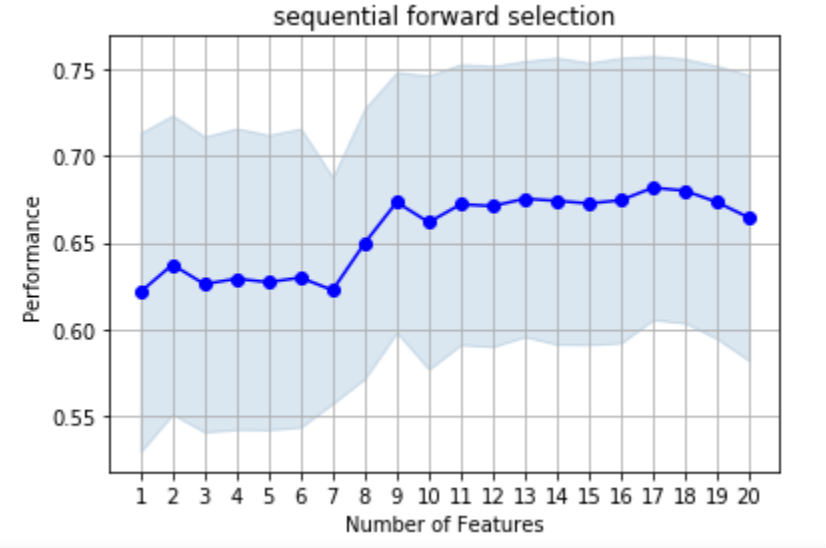

Figure 15: Wrapper method with Random Forest Classifier for 10s timeout dataset

Figure 16 shows that the best performance is achieved with number of features been 6, 9 or 17. The least number of features is 6.

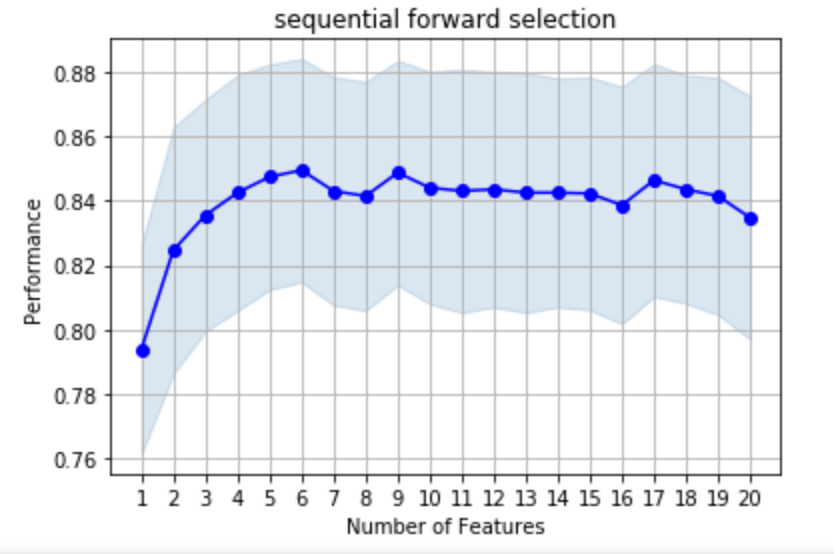

Figure 16: Wrapper method with Random Forest Classifier for 15s timeout dataset

From the 3 graphs, it can be concluded that Random forest can achieve best performance on all three timeout datasets when the number of features equal 9.

Table 6: List of top features with Random forest classifier based on wrapper method

5sec timeout data	10sec timeout data	15set timeout data
Flow Bytes/s	Flow Bytes/s	Flow bytes/s
Flow IAT Mean	Flow IAT Min	Bwd IAT Min
Fwd IAT Mean	Fwd IAT Max	Bwd IAT Mean
Fwd IAT Max	Bwd IAT Std	Flow IAT Std
Bwd IAT Mean	Bwd IAT Min	Active Max
Bwd IAT Max	Active Mean	Idle Min
Bwd IAT Min	Active Std	
Active Max	Active Min	
Idle Mean	Idle Std	
Idle Std	Idle Max	
Flow Duration		

3.9.2.2. Wrapper Method with Decision Tree Classifier

Based on the Decision tree algorithm, the wrapper method gives the graph of how performance varies with the number of features.

From Figure 17, the Decision tree classifier achieves a consistent performance from 4 to 17 features. The performance drops beyond 17 features. The minimum features needed for good performance is 4.

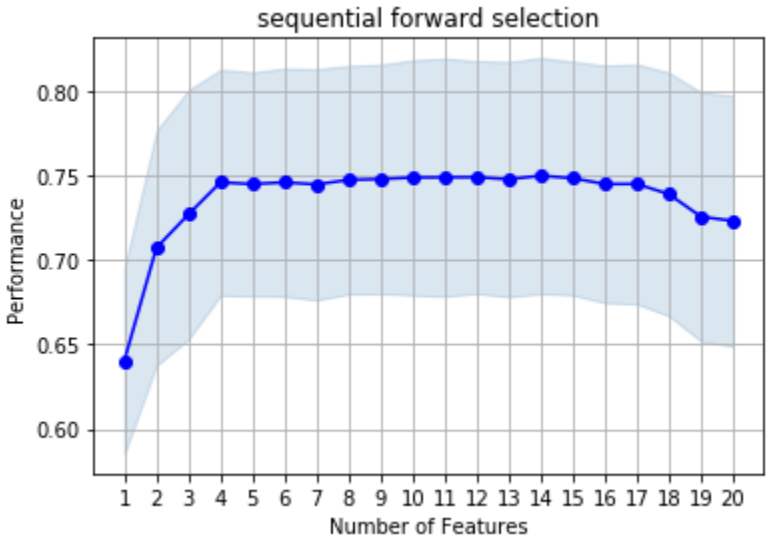

Figure 17: Wrapper method with Decision tree classifier for 5s timeout dataset

Figure 18 also shows a consistent good performance from features 6 through to 20. These 6 features are the minimum number of features that give good performance.

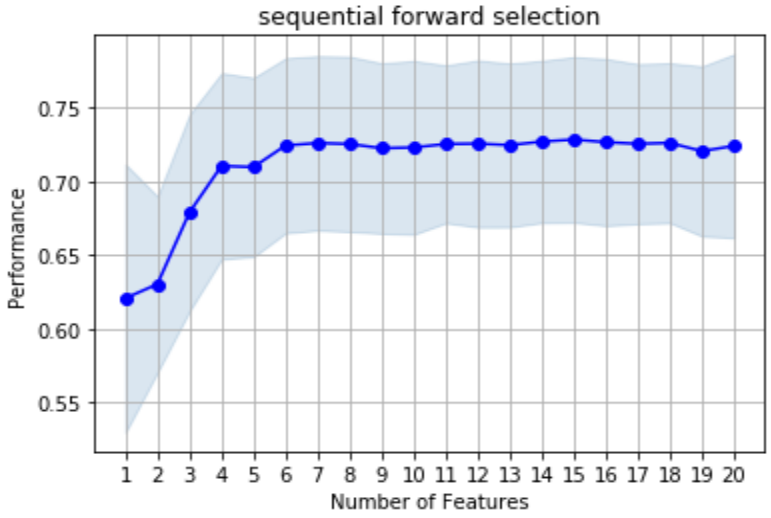

Figure 18: Wrapper method with Decision Tree Classifier for 10s timeout dataset

Figure 19 exhibits the same consistency as seen in Fig 3.10 with the minimum number of features for good performance been 6.

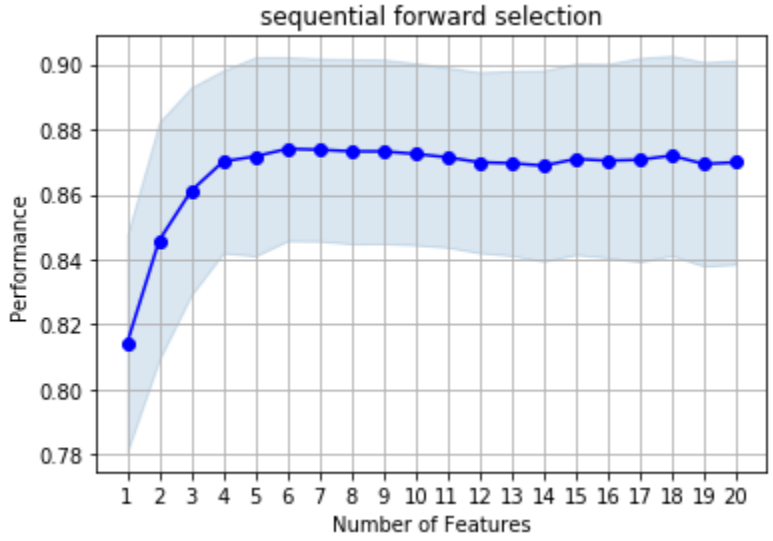

Figure 19: Wrapper method with Decision tree classifier for 15s timeout dataset

From the 3 graphs, with 6 features, Decision tree classifier achieve high performance on all 3 datasets.

Table 7: List of top features with Decision tree classifier based on wrapper method

5sec timeout dataset	10sec timeout dataset	15sec timeout dataset
Flow.Duration	Flow Duration	Bwd IAT Min
Flow Bytes/s	Flow Bytes/s	Bwd IAT Mean
Flow IAT Mean	Flow Packets/s	Flow Bytes/s
Fwd IAT Std	Flow IAT Min	Flow IAT Std
Bwd IAT Mean	Fwd IAT Std	Active Max
Bwd IAT Min	Fwd IAT Min	Idle Min
Active Mean	Idle Std	
Active Max		
Idle Max		
Idle Min		

We also did a feature selection based on feature importance approach called SHAP. SHAP computes the contribution of each to a prediction. It was implemented in python using the 'SHAP' module. SHAP currently works on only tree-based algorithms. We applied this approach on the Random forest and decision tree classifiers. The data was grouped in 10 different classes and the training algorithm was applied on them. The SHAP method then calculated the impact of each feature in all the classes and presents the results in a graph.

3.9.2.3. SHAP method with Decision tree classifier

Figure 20 shows the contribution of each feature on the 5s timeout dataset. Flow bytes/s is seen to have the most impact on the outcome whereas Active Mean, Active Min, Idle Mean, Active Max and Idle Min have small to zero impact on the classification. Using 0.05 SHAP value as cutoff, the contributing features equal 7. They include, Flow Bytes/s, Flow IAT Std, Fwd IAT Min, Bwd IAT Min, Bwd IAT Std, Flow Duration and Flow IAT Min.

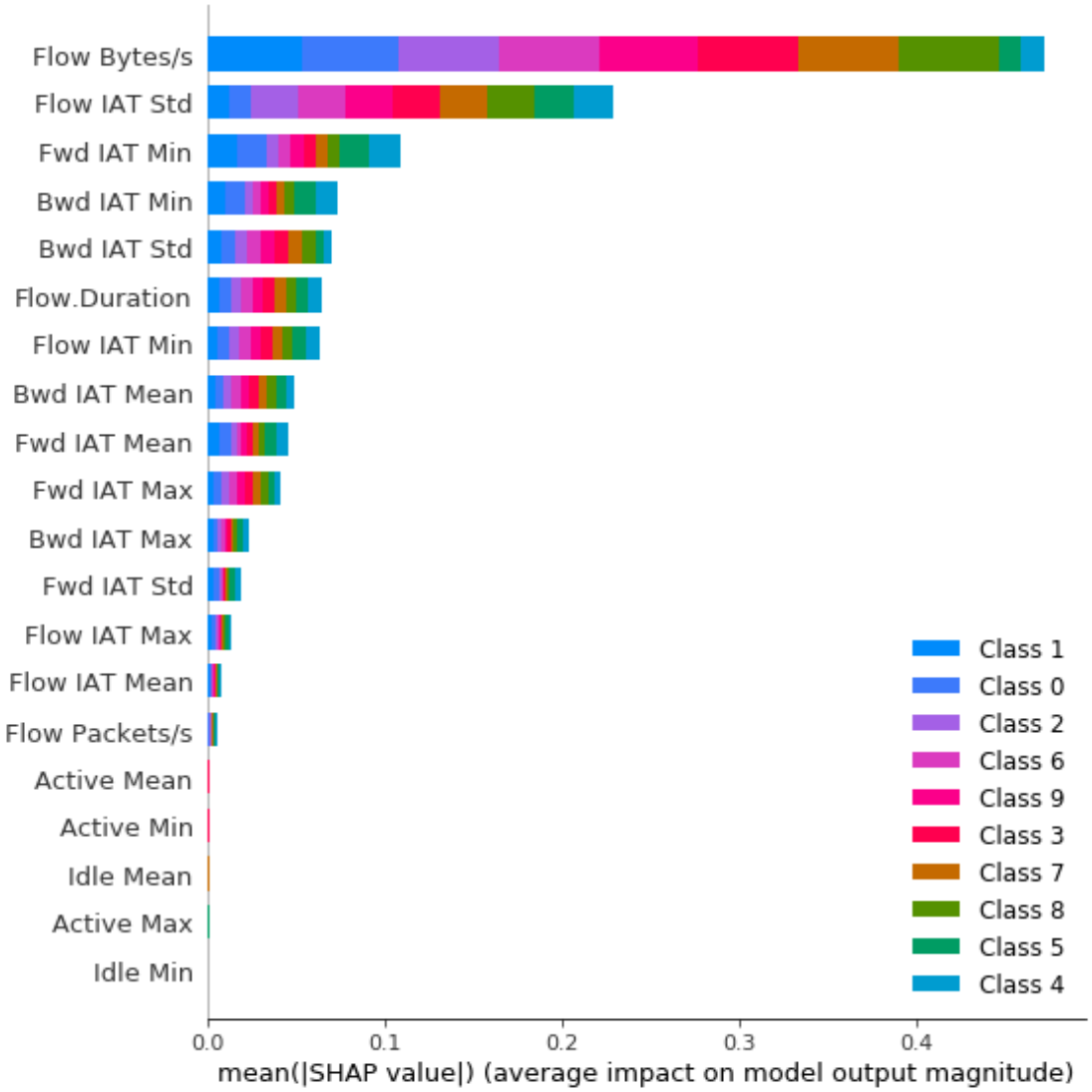

Figure 20: Embedded method SHAP on Decision tree classifier for 5s timeout dataset

On the 10s timeout dataset as shown in Figure 21, using the same cutoff value, the top features are Flow bytes/s, Flow IAT Std, Fwd IAT Std, Fwd IAT Min, Bwd IAT Min, Fwd IAT Mean and Flow IAT Min, a total of 7 features.

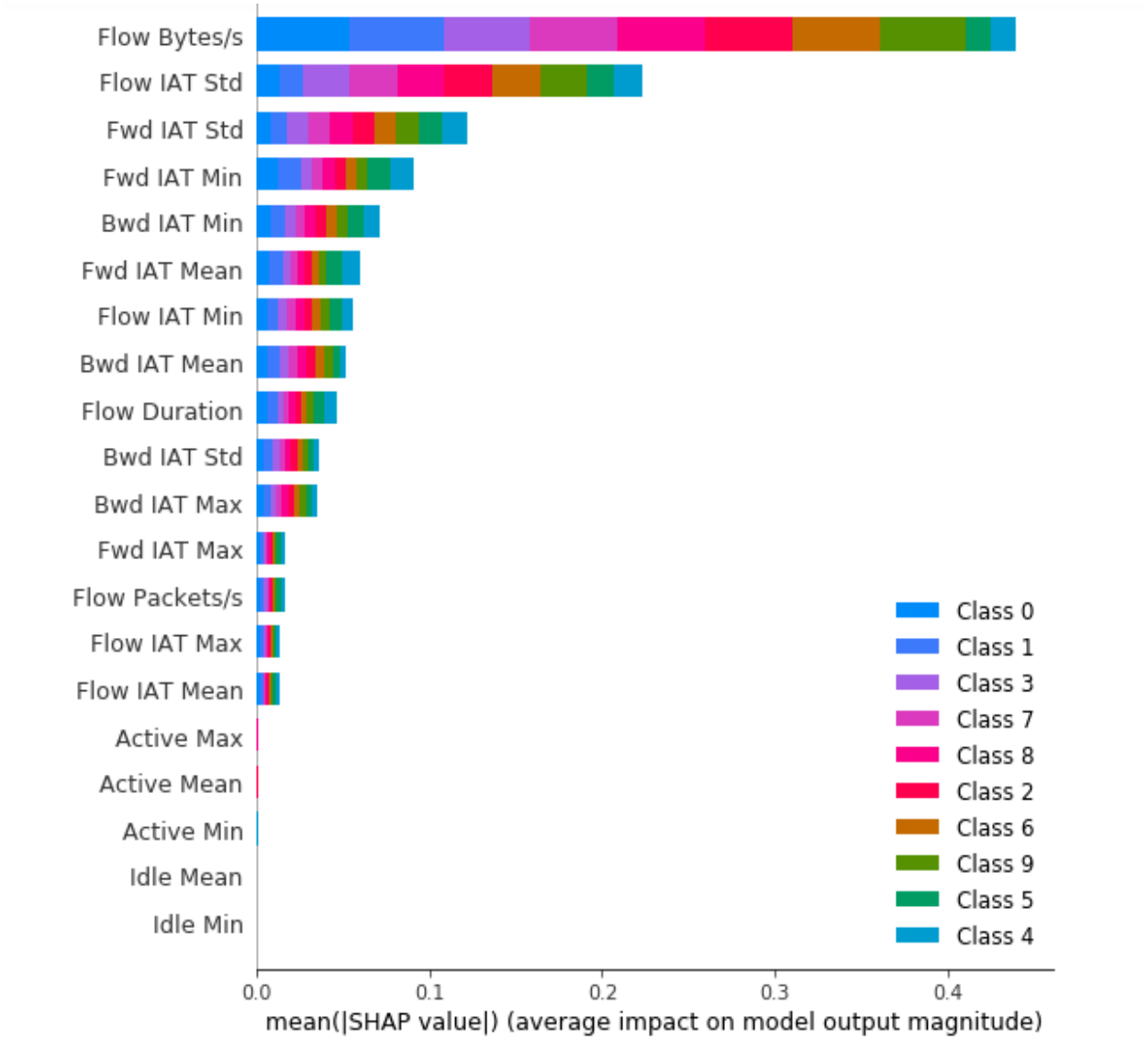

Figure 21: Embedded method SHAP on Decision tree classifier for 10s timeout dataset

In Figure 22, most of the features are seen to have no major impact on the classification. We pick the features that have a SHAP score from 0.05. The total number of such features equal 4. They are Flow bytes/s, Flow IAT Std, Bwd IAT Min and Fwd IAT Min.

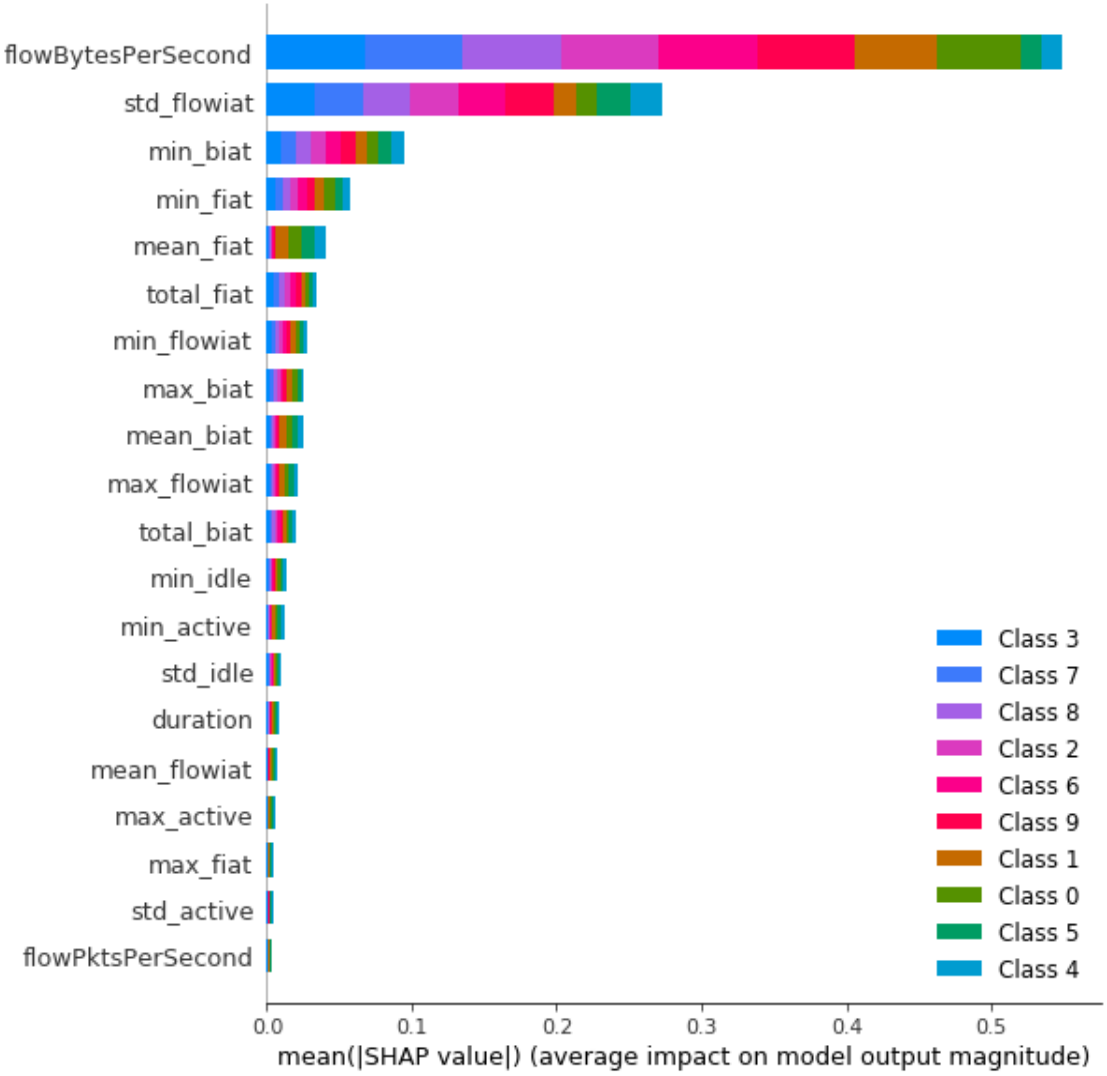

Figure 22: Embedded method SHAP on Decision tree classifier for 15s timeout dataset

From the three graphs, more features contribute to the classification on the 5sec and 10sec timeout datasets as compared to the 15sec timeout datasets. Flow bytes/s is also seen to be a significant contributor to the classification.

3.9.2.4. SHAP Method with Random Forest Classifier

Using the Random forest algorithm, we applied SHAP to determine the contributions of the individual features.

Figure 23 shows the contribution of the features on the 5sec timeout dataset. The Flow Bytes/s feature has the most contribution mean of over 0.16. Majority of the features are seen to have close to equal impact on the classification task. Setting our cutoff mean to 0.05, 10 features are chosen. They are Flow bytes/s, Flow IAT Max, Flow IAT Min, Bwd IAT Mean, Fwd IAT Max, Fwd IAT Std, Fwd IAT Min, Fwd IAT Mean, Flow IAT Mean and Flow packets/s.

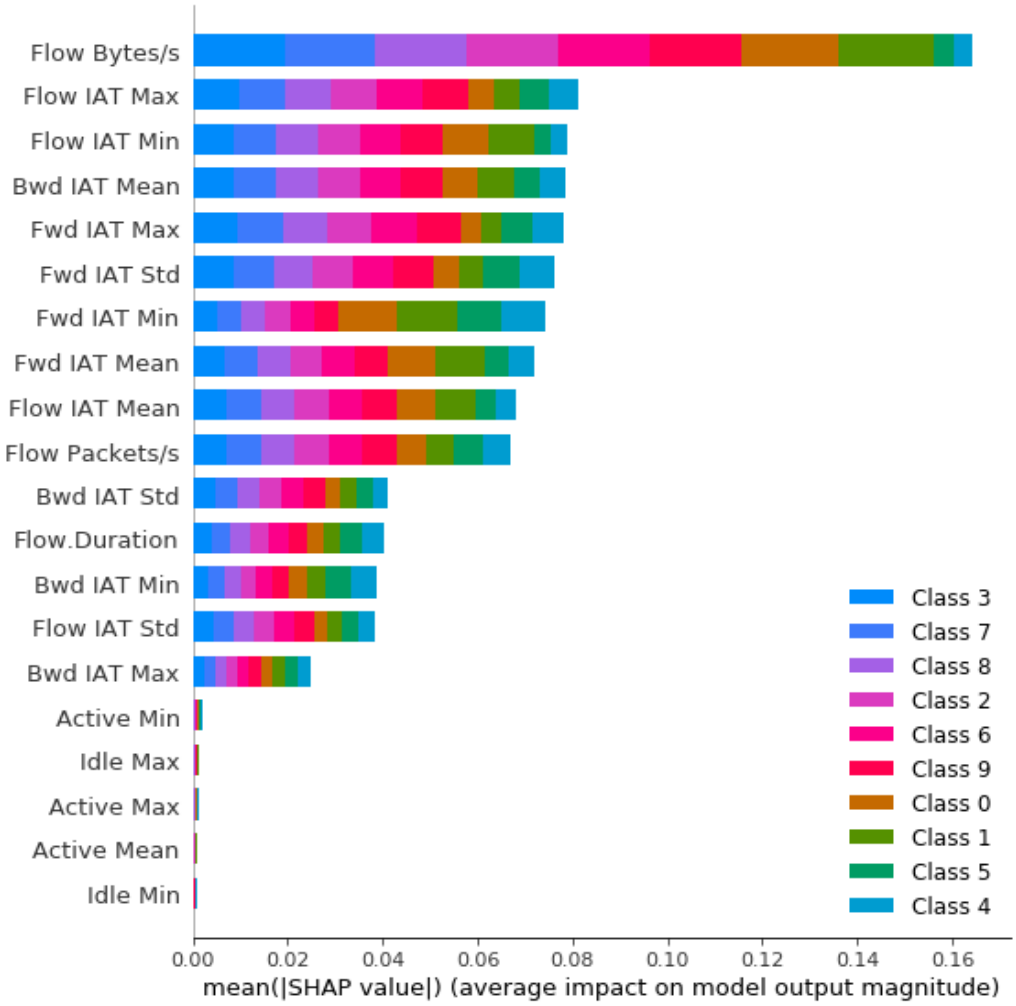

Figure 23: Embedded method SHAP on Random forest classifier for 5s timeout dataset

Applying the same cutoff on Figure 24, we have 10 features which include Flow Bytes/s, Flow IAT Max, Flow IAT Min, Bwd IAT Max, Fwd IAT Max, Fwd IAT Std, Fwd IAT Min, Fwd IAT Mean, Flow IAT Mean and Flow packets/s.

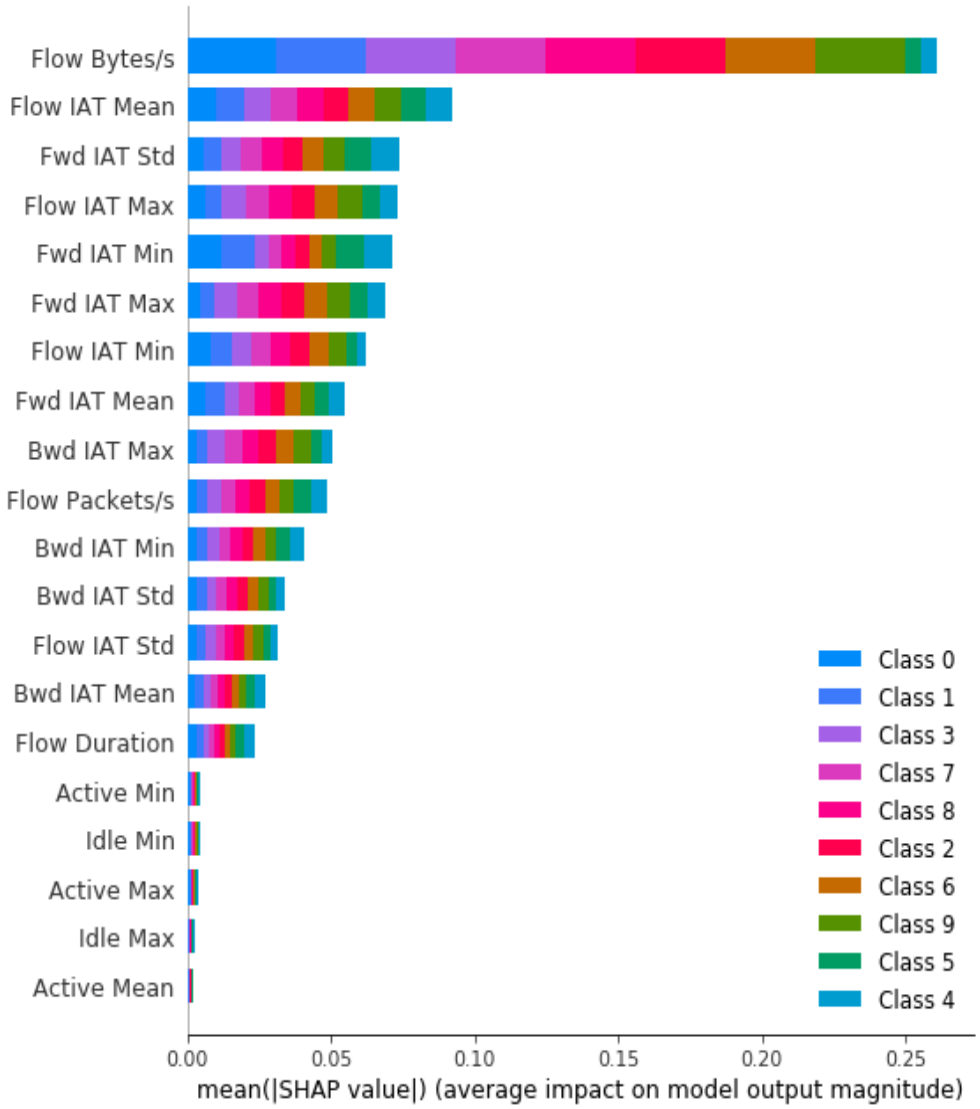

Figure 24: Embedded method SHAP on Random forest classifier for 10s timeout dataset

On the 15sec timeout dataset, 0.05 mean value cutoff gives 8 contributing features as shown in Figure 25. They are Fwd IAT Max, Flow bytes/s, Flow IAT Max, Flow IAT Mean, Bwd IAT Mean, Bwd IAT Min, Flow IAT Std and Flow packets/s.

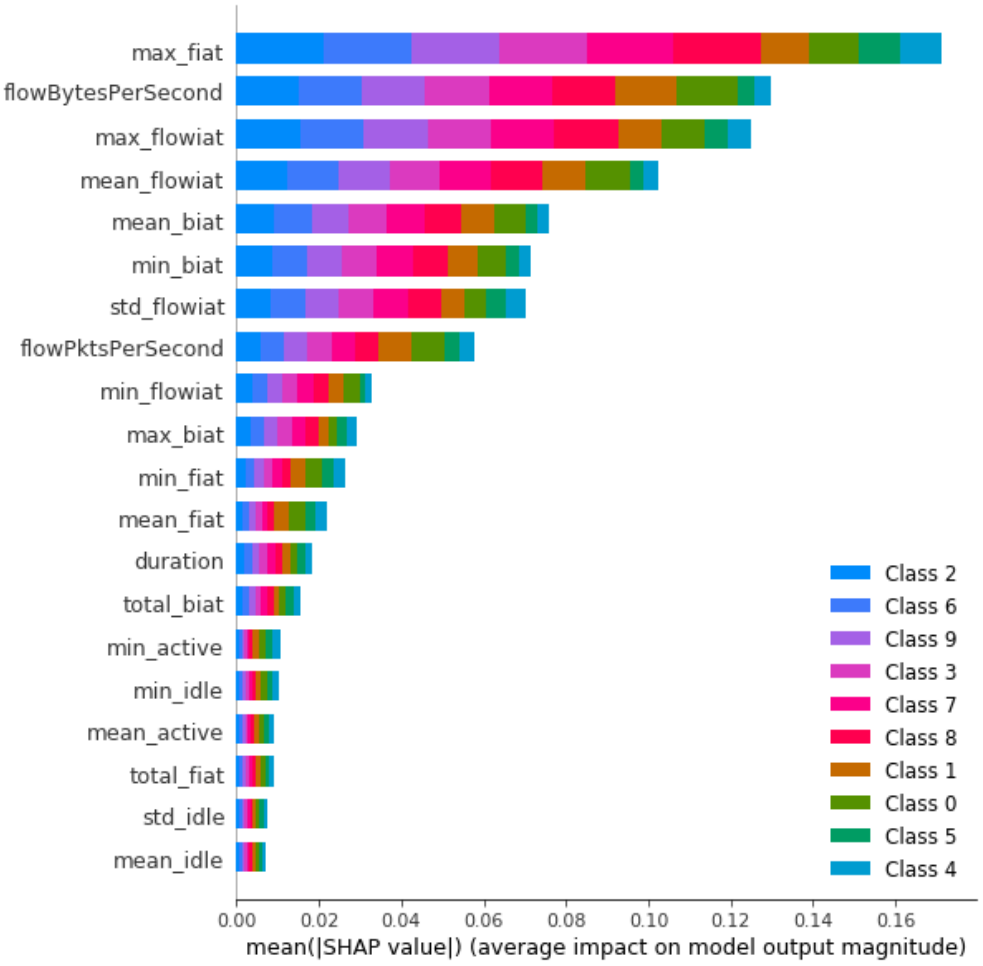

Figure 25: Embedded method SHAP on Random forest classifier for 15s timeout dataset

SHAP method gives a graphical representation of each feature's contribution to the task. Based on a SHAP mean, we have listed a couple of features that significantly impact the prediction for each dataset.

3.9.2.5. Results after Feature Selection

In this section, we apply our algorithms on the datasets based on the features selected in the preceding session. We compare the two feature selection methods based on the accuracies, F1 scores and training times of the algorithms. We also analyze how the different timeouts impact the classification task after feature selection. Finally, we choose the best algorithm, the best set of features and the duration timeout that gives the best classification results.

To compare the different feature selection impact on the different algorithms, we use the average of F1 score and accuracy as well as the training time. An assumed metric termed impact which equals the average of the F1 score, and accuracy is used.

Figure 26 shows the plot of impact and training time against ML algorithm for both feature selection methods on the 5sec timeout dataset. The SHAP feature selection on RFC achieves the highest impact with 0.876. It also records a training time of 0.316sec. From the graph, the RFC algorithm achieves better results in terms of performance (F1-score and Accuracy) whiles DTC algorithm records the lowest training times.

Figure 26: Comparison of feature selection methods on 5sec timeout dataset

Figure 27 also shows the plot of impact and training time against ML algorithm for both feature selection methods on the 10sec timeout dataset. The SHAP feature selection on RFC achieves the highest impact on the 10sec dataset with 0. 861. Both algorithms perform better on the 5sec dataset as compared to the 10sec timeout dataset. DTC with SFS performs better than RFC with SFS unlike in Figure 26. SFS produce smaller training times in both algorithms as compared to SHAP feature selection because SFS result in lesser number of optimal features as compared to SHAP.

Figure 27: Comparison of feature selection methods on 10sec timeout dataset

On Figure 28, RFC with SFS produce the best results. It records an impact score of 0.899 and a training time of 0.05sec. Unlike on the 5sec and 10sec datasets, SFS achieves better results than SHAP with both algorithms.

Figure 28: Comparison of feature selection methods on 15sec timeout dataset

From the three graphs, it is evident that the 15sec timeout datasets achieve better F1 score and accuracy as compared to the other datasets. This concludes that the best timeout for data extraction among the three is 15sec. This is in line with the conclusion made in [40]. From Figure 28, SFS feature selection gives the best results; 0.899 for the RFC algorithm and 0.895 for the DTC method. RFC with SFS has a lower training time compared to DTC with SFS. Though the time difference is not so much, it is very relevant as the size of data increases. From these results, we choose Random forest classifier as our preferred ML algorithm and the SFS feature selection method. This approach achieves an accuracy of 0.883 and an F1 score of 0.914 (precision=0.927 and recall=0.902). In an imbalance data as in our case, the best metric for assessment is F1 score and an F1 score of 0.914 in traffic classification is well within acceptable limits.

Figure 29:Comparison of RFC with and without feature selection

Figure 29 compares RFC on the 15sec timeout dataset with and without feature selection. Feature selection increases the accuracy and F1 score to 0.883 and 0.914 respectively. It also reduces the number of features from 23 to 6. With 6 features, the VNF will take lesser time computing these features from extracted flow. The training time decreases from 0.07 to 0.05sec which is very vital for real time application and as the data size increases. The

features used are Flow bytes/s, Bwd IAT Min, Bwd IAT Mean, Flow IAT Std, Active Max and Idle Min. The graph in Figure 29 shows that feature selection improves the performance metrics of a ML algorithm.

Table 8 below shows a comparison of our results to other previous research works.

Table 8: Comparison of our results to other research works

Our work	Work presented in [40]	Work presented in [38]
We achieve a precision score of 0.927 on the 15sec timeout dataset.	They achieved the best precision score of 0.836 on the 15sec timeout dataset.	They achieved a precision score of 0.87 by applying DNN on the 15sec timeout dataset.
After SFS feature selection, we reduced our number of features to 6 and achieved best results.	The best results were achieved using Infogain and Ranker as feature selection approach and reduced features to 15.	They did not apply feature selection.

3.10. Conclusion

In this chapter, we have presented the approach and dataset used for our experimentation. We also selected three different ML algorithms, applied them on our datasets and presented the results. We adopted two major feature selection methods and compared them on the two top ML algorithms and the datasets. We also investigated the impact of the flow timeout on traffic classification. We concluded that 15sec timeout achieves the best classification results. We have chosen Random Forest classifier with SFS feature selection as our best solution. We also presented a comparison between our best classifier with and without feature selection. In conclusion, we compared our results to that achieved by fellow researchers on the same dataset.

4.0. QoS Requirement Based Optimized Routing

In a central controlled SDN architecture, OpenFlow protocol allows the controller to periodically collect information from the switches. Our architecture leverages the OpenFlow protocol to collect network statistics information and traffic matrices.

When the SDN controller receives a PACKET_In request for a flow, it calculates the route of the packet based on its global view of the network. It then installs the flow rules on the switches for routing whiles it awaits the result of the classification task. After the classification by the VNF, the controller receives the packet with the QoS requirements indicated. The controller samples the current network state (NS) and traffic matrices (TMs). The controller uses the NS, TMs and the QoS requirements as input to the trained ML algorithm to identify the optimal routing path. It then installs these flow rules on all the switches in the calculated path with high priority. Since these new flow rules have higher priority than the previously installed rules, the switches will route traffic through the newly installed flow rules.

In this chapter, we look at providing optimized routing based on identified QoS requirement of a traffic flow. We utilize a Deep neural network to identify the routing policy that will provide the QoS requirements of a traffic flow based on its QoS class. We present the approach, the neural network architecture, dataset and data preparation as well as evaluation results.

4.1. Approach

The traffic classifier presented in the previous chapter utilizes the statistical features of a flow to identify the latency and bandwidth requirements of a flow. These requirements are classified as high, mid or low. We utilize the latency requirements of the flow to identify optimal routes.

Table 9: Latency requirements

Latency class	Numerical latency requirement
Low latency/real-time traffic	10-30ms
Mid latency	30-60ms
High latency/best effort	Above 60ms

Our machine learning model combines the traffic matrices of the network, the network states and the desired latency to give the best routing policy.

4.2. Dataset

The dataset utilized for this section is based on the NSFNET topology. It is a 14-node network mostly used in the research community as a benchmark for routing solutions. We used the datasets provided by [48].

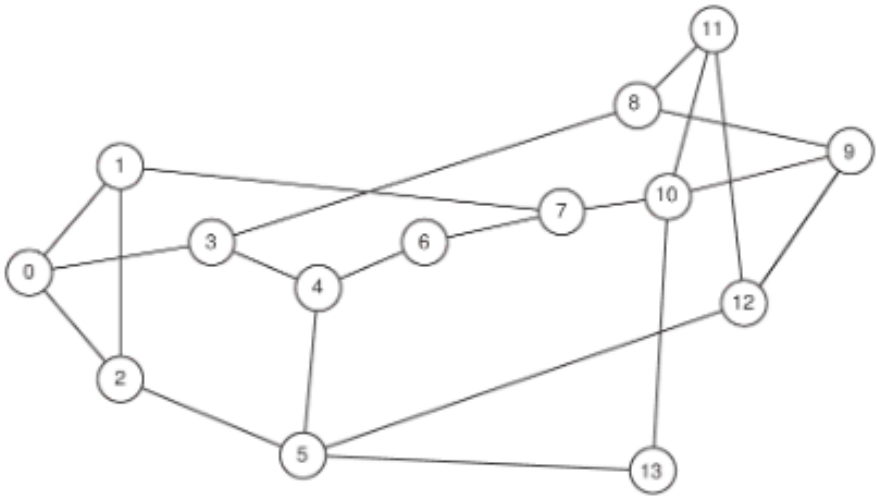

Figure 30: 14-node NSFNET topology

They acquired the data by using a custom-built packet-level simulator in OMNET++ to measure end to end delays in the NSFNET topology for different combinations of routing schemes and traffic matrices with different traffic intensity. The dataset consists of 250,000 samples.

The datasets consist of different matrices describing the Routing information base (RIB) at each node. There about 174 different routing policies. These policies are based on popular routing algorithms and heuristic algorithms. They simulated the 14-node network to run on each of the 174 routing policies and combined with different network conditions. They then measured the end to end delays for each routing policy and network condition combination. Our goal is reverse engineer their findings to train a model that will propose a routing policy based on the network condition and the required latency (end-to-end delay) of the flow.

4.3. Data Preparation

The input data consists of 1765 columns as provided by [48]. Each instance/row represents a unique network condition (traffic matrix). The dataset consisted of features like:

- Bandwidth (in kbps) transmitted in each source-destination pair in the network (in both directions).
- Absolute number of packets transmitted in each source-destination pair (in both directions).
- Average per-packet delay over the packets transmitted in each source-destination pair.
- Variance of the per-packet delay (jitter) over the packets transmitted in each source-destination pair.

The output data consisted to corresponding routing policy (RIB) for each network condition. The output RIB for each instance was a 14×14 matrix. We assigned IDs to each of these outputs. Assigning IDs makes it computationally bearing compared to a 14×14 output. Our dataset has 250000 different network scenarios hence the input matrix is 250,000 × 1765 and an output matrix of 250,000×1 representing the ID assigned to the corresponding routing policy.

The features values were normalized into a value between 0 and 1 for efficient training of the neural network. This was achieved using the MinMaxScaler provided by the Scikit-learn library. The dataset was split into 70% training and 30% testing samples.

4.4. Deep Neural Network

A Deep Neural Networks (DNN) is a NN with multiple hidden layers (usually at least two hidden layers) of units between the input and output layers [49]. DNNs have gained a lot success in common tasks like image classification, language translation and other major ML problems. Computer network researches have also adopted DNNs to improve network traffic engineering. In this study we used the feed forward deep neural network with back propagation learning algorithm. In a feed forward network, information flows from the input nodes through the network to the output nodes without any feedback/loop connections.

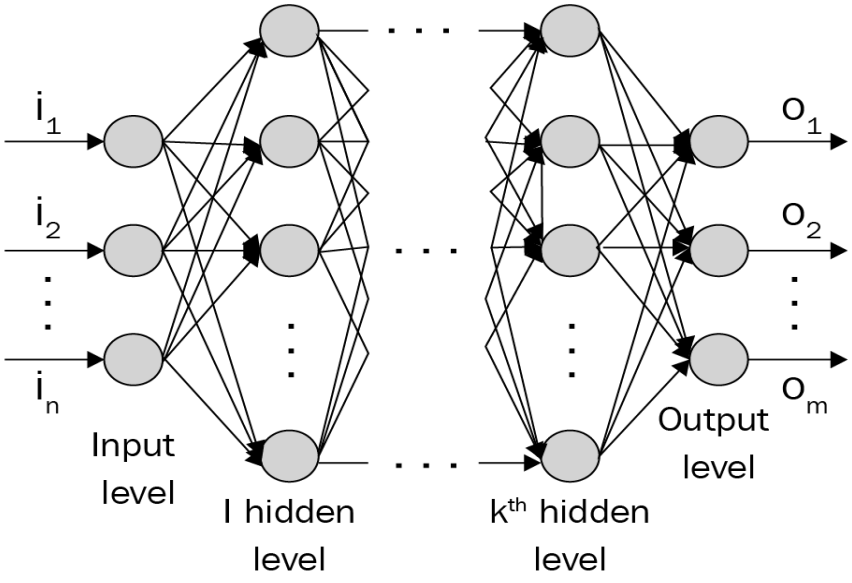

Figure 31: Deep feed forward neural network

4.6. Back Propagation Learning Algorithm

Back propagation is the most commonly used learning algorithm for feed forward neural networks. The algorithm uses gradient descent optimization method. The algorithm learns by matching an input data/instance (traffic matrix, network state and delay requirement) to the output (routing policy ID) by building a function. It compares the prediction to the correct answer and propagates the error at the output node to the inner nodes. The learning process is supervised.

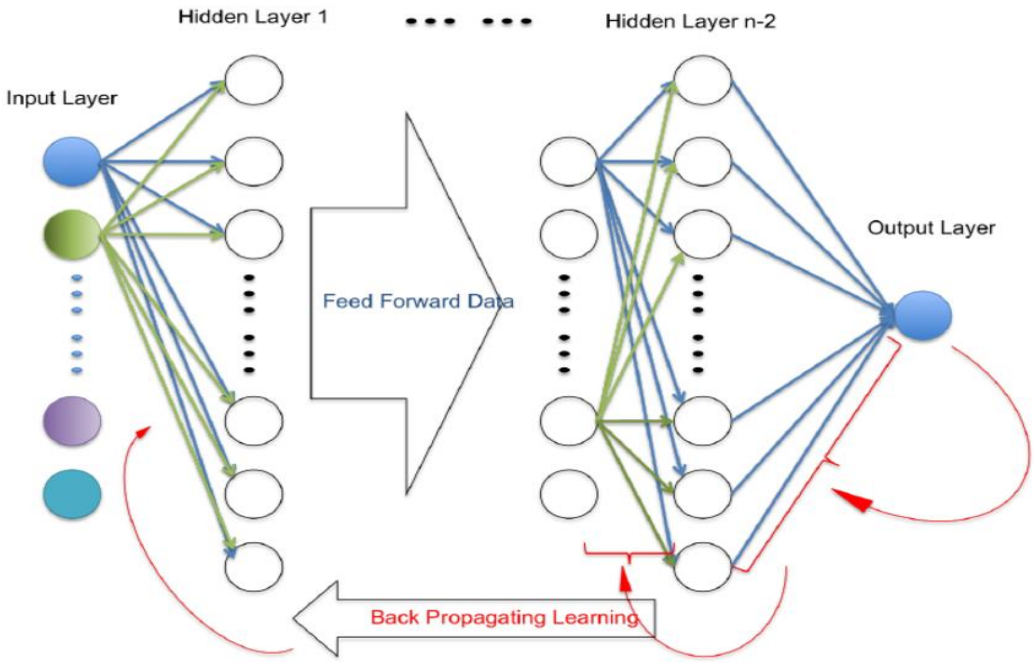

Figure 32: Back propagation process

The network then modifies the weights for each of the neurons based on the propagated error. The forward and back process is repeated until the error is reduced to acceptable value.

4.8. The Neural Network Architecture

To build our model, we utilize the 1765 features of the dataset. The neural network was configured as follows:

1. **Input layer**: The input layer consists 50 nodes and an input dimension equal the size of the input instance dimension. It uses the ‘relu’ activation.
2. **Output layer**: The output layer uses the ‘softmax’ activation function which is commonly used for multi-class outputs as our work.
3. **Hidden layers**: We experimented for multiple hidden layers (2,3,4 and 5) and compared the results in terms of training time and accuracy. All hidden layers use the ‘relu’ activation.
4. **Optimizer**: We use the ‘Adam optimizer’ which is widely used in most deep learning applications. It leverages the power of adaptive learning rates methods to find individual learning rates for each parameter [50].
5. **Learning rate**: The learning rate is the default learning rate of the ‘Adam Optimizer’ which is 0.001.
6. **Loss**: ‘Categorical crossentropy’ is used. It is used for single label categorization i.e. when each instance can belong to only one class (routing policy).
7. **Epochs**: The model was trained through 30 epochs. An epoch represents a complete training over the training dataset.
8. **Metrics**: Metric used is Accuracy. Accuracy indicates the number of correctly classified instances.
9. **Validation data**: A validation split of 0.3 was used. 30% of the training data was used for validation.

4.9. Evaluation

This section presents the experimental process and the results achieved.

We first analyze the results obtained from testing different hidden layers in building the neural network architecture. We review the loss and training time against the number of hidden layers. Secondly, we conclude on a model and presents its accuracy over different training epochs. Finally, we present the results on the chosen model on the test data set.

The Neural Network model was implemented in the PyCharm IDE[51] using the Keras Library[56]. The same computer described in the previous contribution was used.

Several simulations were run to identify the number of hidden layers that will give the best results in terms of training time and accuracy.

Figure 33: Loss against the number of hidden layers in our model

Figure above shows how the loss varies with the number of hidden layers. The number of hidden layers of a model is problem dependent. It is seen that the loss starts to increase after 3 hidden layers. Using hidden layers more than the sufficient/needed number of layers can cause the accuracy of the model on test data to decrease. The model will overfit to the training data and perform poorly to the test data. From the graph, 2 or 3 hidden layers give good losses compared to the rest.

Figure below also compares the training time to the number of hidden layers. It can be observed that the training time increases as the number of hidden layers increase. In our work, we used the same number of neurons (100) for each hidden layer so as the number of hidden layers increase, so does the number of neurons. Since each neuron computes its weight, the training time increases as our number of layers increase.

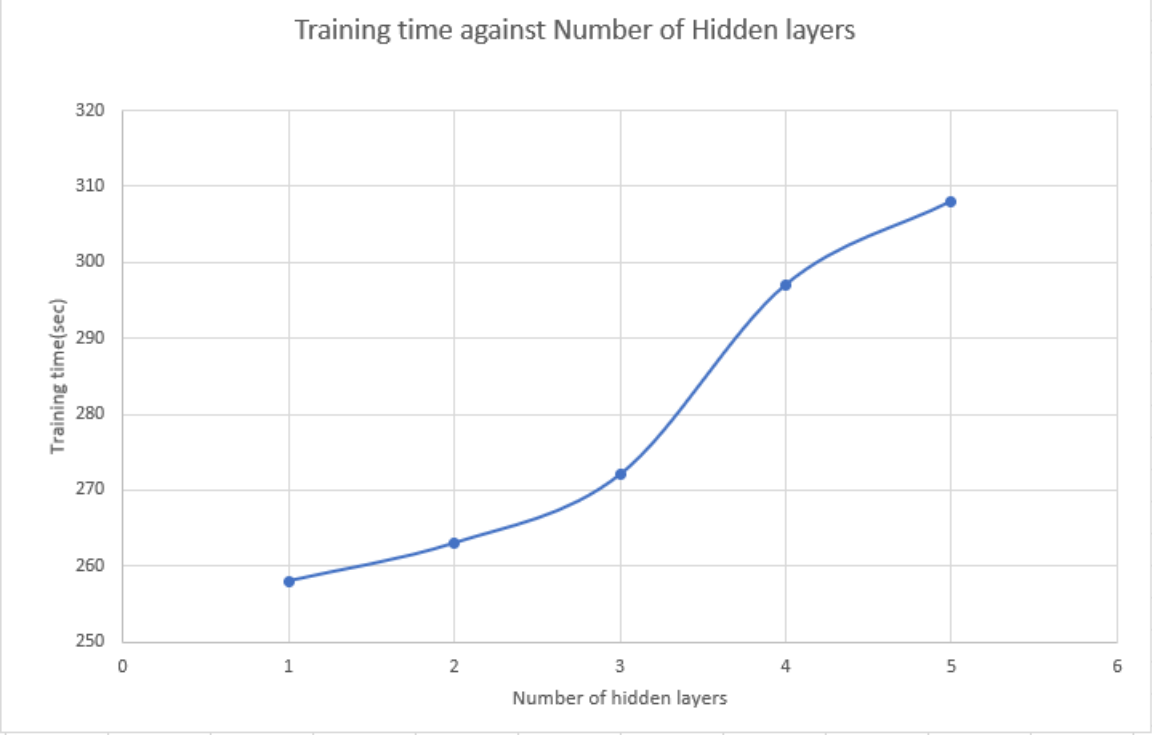

Figure 34: Training time against the number of hidden layers of our model

From Figure 33, both 2 and 3 hidden layers give the lowest loss values. On the time curve in Figure 34, a model with 2 hidden layers has lesser training time than 3 hidden layers. We therefore chose the model with 2 hidden layers since we would like to build a model for real time use hence the need for small training time.

Choosing a model with 2 hidden layers, we trained the model on the training dataset and used 30% of the dataset for validation over 30 training epochs. Figure 35 below shows how the loss and accuracy perform over the training epochs. It can be observed that the trained model fits quite well on the validation data.

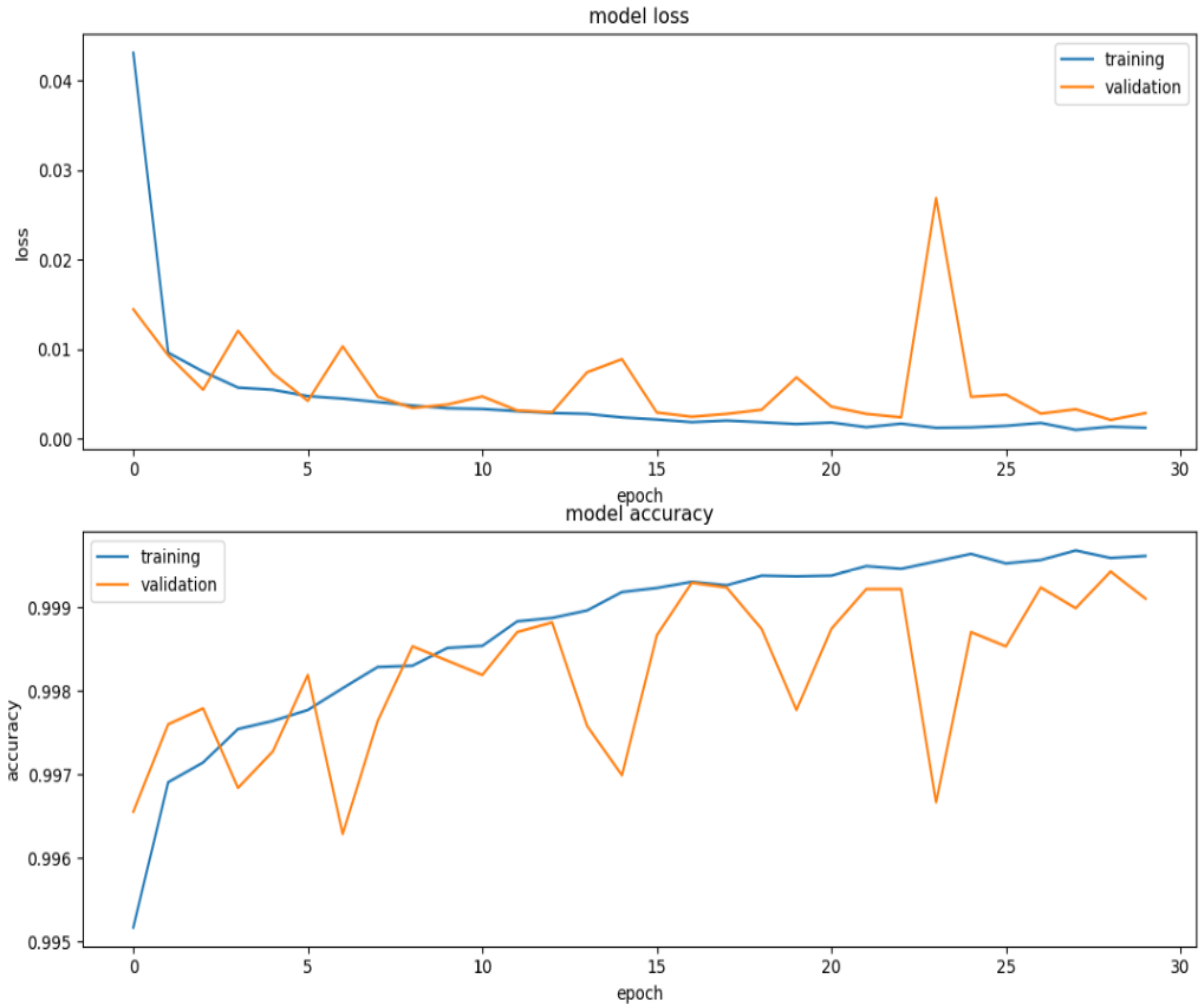

Figure 35: Accuracy and loss over training epochs

Figure 36 explicitly shows the accuracy over training epochs on the training data. The model achieves better accuracy as the number of epochs increases. The curve fitting the training data smoothens after 25 epochs. The accuracy values do not change significantly after that. Training for the 30 epochs gives the model that can give optimal routing solution. Each training epoch takes about 10s on our system and hence the model takes 5mins to train.

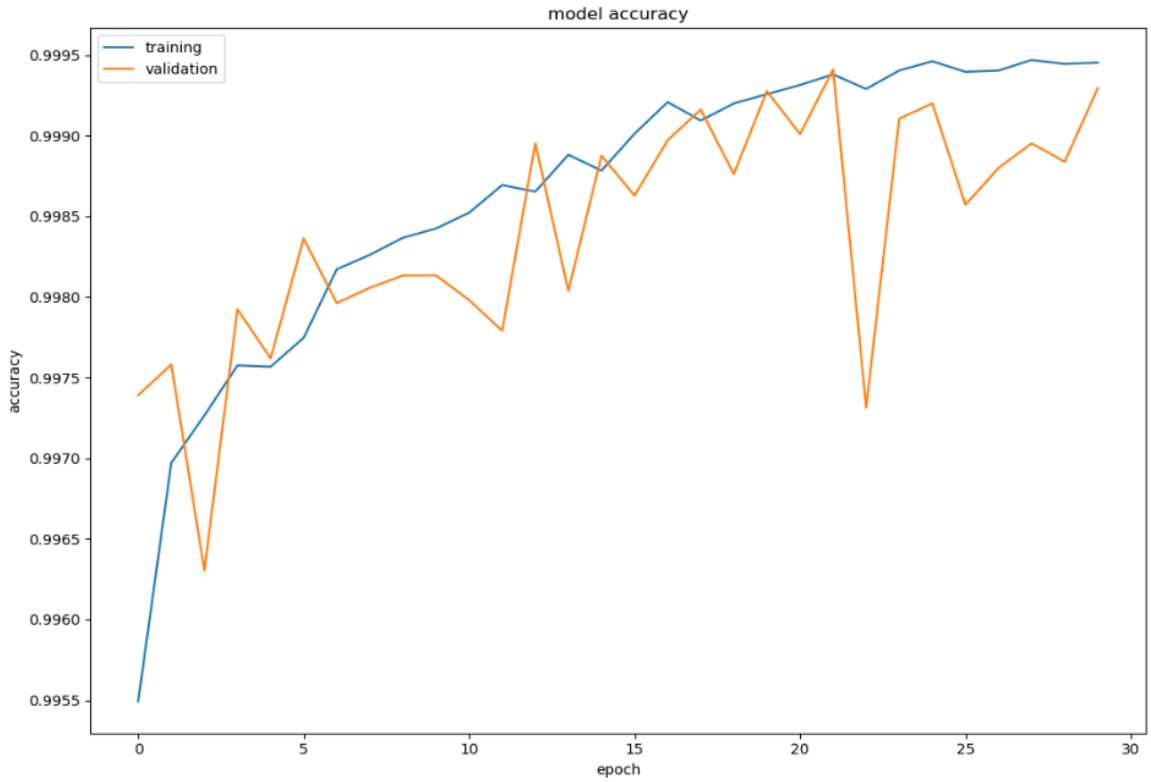

Figure 36: MSR over training epochs

Finally, we applied the trained model on the test data (unseen data). A graph displaying showing the accuracies on the model on the training, validation and test data is presented in Figure 37. On the unseen data, the model performs well by achieving an accuracy of 0.9995. This confirms that the model did not underfit or overfit on the training data. The model can predict the optimal routing policy in 135μs compared to heuristic algorithms that can take up to 120ms to calculate the path.

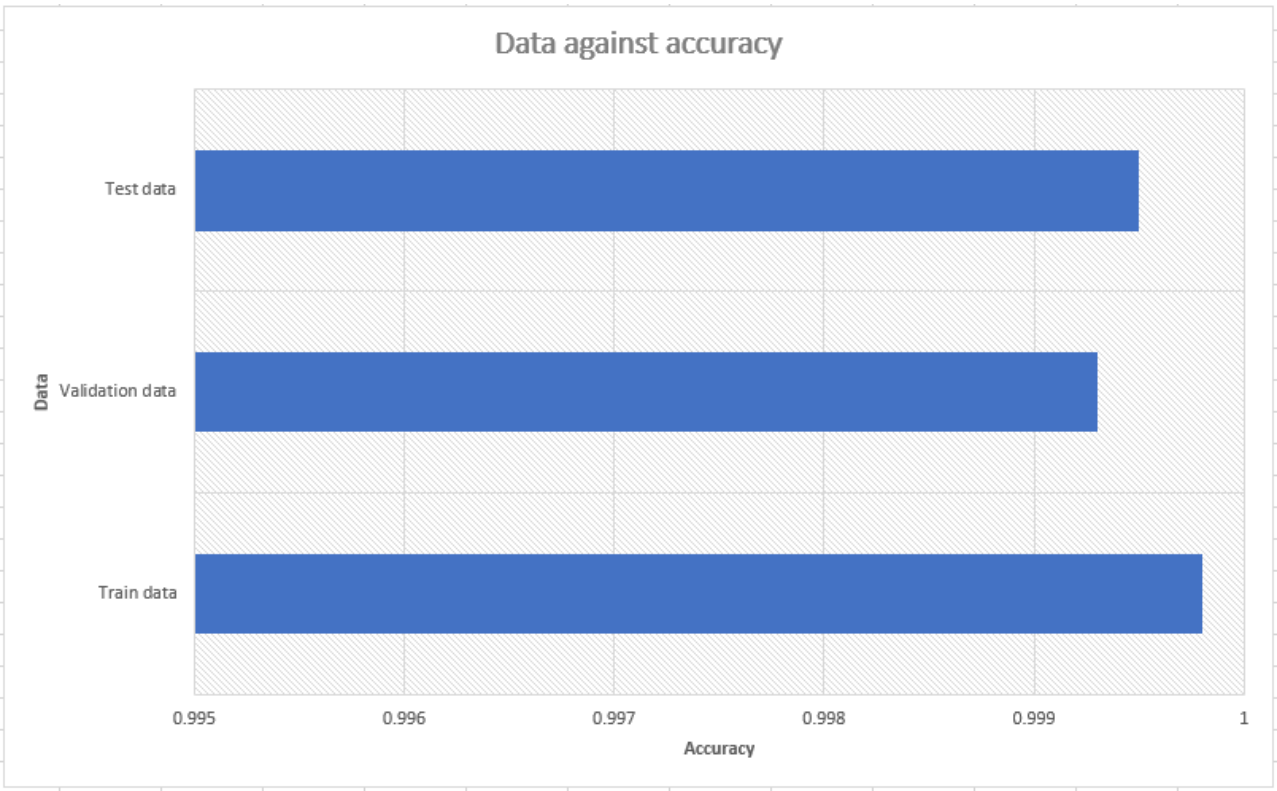

Figure 37: Accuracy of the model on different data

Two major works regarding supervised learning for route optimization in SDN are presented in [32] and [33]. In [33], they achieved a Neural network accuracy of 83.2% on the NSFNeT topology based on their approach. Our approach achieves a 99.95% in comparison. Also, in [33], they built a model for each source-destination pair which is impractical as the size of the network increases. [32] on the other presented a model that gave a heuristic-like routing solution when given the network state and traffic matrix as input. Their approach did not give emphasis to the QoS requirements of the flow in question. We added the QoS(latency) requirement of the flow to the input making the routing solution flow dependent instead of a general solution depending on the network condition.

4.10. Conclusion

In this chapter, we have presented the approach and dataset used for our experimentation on route optimization. We used a deep neural network with back propagation learning algorithm. We trained the model for different number of hidden layers and chose 2 hidden layers as the optimal model considering accuracy and training time. We presented a graph of accuracy and loss against training epochs for the model. The model achieved a 0.9995 accuracy and an entropy loss of 0.000497 on the test data. We also compared our approach and results to some works done.

5.0. Conclusion and Future Work

In this chapter, we present a summary of our work and outline the future works in line with our contributions yet to be explored. Section 5.1 discusses the conclusions based on the results from Chapter 3 and 4. Section 5.2 describes our recommendations for future work.

5.1. Summary of Work Done

We started of our work by pointing out the need to integrate IoT and SDN to provide a framework capable of meeting the growing demands on IoT devices. An SDN-IoT framework leverages the centralized architecture of SDN to provide a platform to incorporate ML for traffic engineering. We provided a review of Software Defined Networking and Machine Learning. We also presented literature on some Machine Learning algorithms.

We provided a system design that utilizes SDN and VNF to allow ML application for traffic classification and route optimization. Three Machine Learning algorithms- Random Forest Classifier, K-Nearest Neighbors and Decision Tree Classifier are applied on a ToR dataset for classification. The ToR dataset is an encrypted dataset. The ML learning models use the statistical features of the traffic data to classify the traffic into QoS requirements; latency and bandwidth. Random Forest Classifier and Decision Tree Classifier performed best for both F1 score and training time. K-Nearest Neighbors was not considered further due to its low scores compared to the other 2. Two Feature selection methods—SHAP and SFS were applied on the 2 chosen ML algorithms to improve their performance and reduce training time but reducing the number of features. We found that Random forest classifier with SFS feature selection produced the best performance. It attained accuracy and F1 score of 0.883 and 0.914 respectively by using 6 features out of 23.

The research also investigated the flow timeouts that gave the best data for traffic classifications. The different ML algorithms were applied on datasets corresponding to

three different flow timeout values: 5s, 10s and 15s. It was found that the 15s timeout dataset gave the best classification results.

Finally, we presented a Deep neural network model that uses the latency requirement of the classified traffic, the network state and network traffic matrix to determine the optimal route for the flow. We used dataset derived from OMNET++ simulations of the 14-node NSFNET topology. The deep neural network consisted of 2 hidden layers and achieved an accuracy of 0.9995 on the test data.

5.2. Future Work

We outline below suggestions for our future work.

- Test on Other traffic datasets

  Performance is expected to change for different traffic datasets. It would be useful to test how accurate the same ML algorithms with the chosen features will perform on different traffic dataset. Results could be compared when the ML algorithms are trained on the new datasets and when they are trained on the current dataset then tested on the new network.

- Test of different network topology

  The optimized route model can be tested on other network topologies to confirm its performance. Further investigations can be carried to see how the model will perform in instances where there are link failures.

- Test with Additional Algorithms

  Testing with other ML algorithms or performing parameter tuning on the tested algorithms could produce better results for either the measured accuracy or runtime in the traffic classification problem.

- The proposed 2-step traffic engineering design can be simulated to compare its feasibility in terms of computation time and scalability.

References

[1] Hassan, Q. F. (Ed.). (2018). *Internet of Things a to Z: Technologies and Applications*. John Wiley & Sons.

[2] Manyika, J., Chui, M., Bisson, P., Woetzel, J., Dobbs, R., Bughin, J., & Aharon, D. (2015). Unlocking the Potential of the Internet of Things. *McKinsey Global Institute*.

[3] Farhady, H., Lee, H., & Nakao, A. (2015). Software-defined networking: A survey. *Computer Networks*, *81*, 79-95.

[4] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., & Turner, J. (2008). OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, *38*(2), 69-74.

[5] Xie, J., Yu, F. R., Huang, T., Xie, R., Liu, J., Wang, C., & Liu, Y. (2018). A survey of machine learning techniques applied to software defined networking (SDN): Research issues and challenges. *IEEE Communications Surveys & Tutorials*, *21*(1), 393-430.

[6] Wu, D., Arkhipov, D. I., Asmare, E., Qin, Z., & McCann, J. A. (2015). UbiFlow: Mobility management in urban-scale software defined IoT. In *2015 IEEE Conference on Computer Communications (INFOCOM)*, pp. 208-216.

[7] Kreutz, D., Ramos, F., Verissimo, P., Rothenberg, C. E., Azodolmolky, S., & Uhlig, S. (2014). Software-defined networking: A comprehensive survey. *arXiv preprint arXiv:1406.0440*.

[8] Goransson, P., Black, C., & Culver, T. (2016). *Software defined networks: a comprehensive approach*. Morgan Kaufmann.

[9] Dargahi, T., Caponi, A., Ambrosin, M., Bianchi, G., & Conti, M. (2017). A survey on the security of stateful SDN data planes. *IEEE Communications Surveys & Tutorials*, *19*(3), 1701-1725.

[10] Lange, S., Gebert, S., Zinner, T., Tran-Gia, P., Hock, D., Jarschel, M., & Hoffmann, M. (2015). Heuristic approaches to the controller placement problem in large scale SDN networks. *IEEE Trans. on Network and Service Management*, *12*(1), 4-17.

[11] Hu, Y., Wendong, W., Gong, X., Que, X., & Shiduan, C. (2013). Reliability-aware controller placement for software-defined networks. In *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pp. 672-675.

[12] *Machine Learning*. (n.d.). Retrieved from https://en.wikipedia.org/wiki/Machine_learning.

[13] Han, J., Pei, J., & Kamber, M. (2011). *Data mining: concepts and techniques*. Elsevier.

[14] Kubat, M. (2017). *An introduction to machine learning* (Vol. 2). Cham, Switzerland: Springer International Publishing.

[15] François-Lavet, V., Henderson, P., Islam, R., Bellemare, M. G., & Pineau, J. (2018). An introduction to deep reinforcement learning. *Foundations and Trends® in Machine Learning*, *11*(3-4), 219-354.

[16] Bishop, C. M. (2006). *Pattern recognition and machine learning*. springer.

[17] *Feature Engineering*. (n.d.). Retrieved from https://en.wikipedia.org/wiki/Feature_engineering.

[18] Dietterich, T. G. (2000). Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pp. 1-15. Springer, Berlin, Heidelberg.

[19] *Tree* (n.d.) Retrieved from scikitlearn.org/stable/modules/tree.html.

[20] Nguyen, T. T., & Armitage, G. J. (2008). A survey of techniques for internet traffic classification using machine learning. *IEEE Communications Surveys and Tutorials*, *10*(1-4), 56-76.

[21] Moore, A. W., & Zuev, D. (2005). Internet traffic classification using bayesian analysis techniques. In *ACM SIGMETRICS Performance Evaluation Review*, Vol. 33, No. 1, pp. 50-60.

[22] Xiao, P., Qu, W., Qi, H., Xu, Y., & Li, Z. (2015). An efficient elephant flow detection with cost-sensitive in SDN. In *2015 International Conference on Industrial Networks and Intelligent Systems (INISCom)*, pp. 24-28.

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

[24] Li, Y., & Li, J. (2014). MultiClassifier: A combination of DPI and ML for application-layer classification in SDN. In *The 2014 IEEE International Conference on Systems and Informatics (ICSAI)*, pp. 682-686.

[25] Rossi, D., & Valenti, S. (2010). Fine-grained traffic classification with netflow data. In *Proceedings of the 6th international wireless communications and mobile computing conference*, pp. 479-483.

[26] Qazi, Z. A., Lee, J., Jin, T., Bellala, G., Arndt, M., & Noubir, G. (2013). Application-awareness in SDN. In *ACM SIGCOMM computer communication review*, Vol. 43, No. 4, pp. 487-488.

[27] Erman, J., Arlitt, M., & Mahanti, A. (2006). Traffic classification using clustering algorithms. In *Proceedings of the 2006 SIGCOMM workshop on Mining network data*, pp. 281-286.

[28] McGregor, A., Hall, M., Lorier, P., & Brunskill, J. (2004). Flow clustering using machine learning techniques. In *International workshop on passive and active network measurement*, pp. 205-214. Springer, Berlin, Heidelberg.

[29] Wang, P., Lin, S. C., & Luo, M. (2016). A framework for QoS-aware traffic classification using semi-supervised machine learning in SDNs. In *2016 IEEE International Conference on Services Computing (SCC)*, pp. 760-765.

[30] Yu, C., Lan, J., Xie, J., & Hu, Y. (2018). QoS-aware traffic classification architecture using machine learning and deep packet inspection in SDNs. *Procedia Computer Science*, *131*, 1209-1216.

[31] Rouse, M. (n.d.). *Virtual Network Functions.* Retrieved from https://searchnetworking.techtarget.com/definition/virtual-network-functions-VNF

[32] Azzouni, A., Boutaba, R., & Pujolle, G. (2017). NeuRoute: Predictive dynamic routing for software-defined networks. In *2017 IEEE International Conference on Network and Service Management (CNSM)*, pp. 1-6.

[33] Yanjun, L., Xiaobo, L., & Osamu, Y. (2014). Traffic engineering framework with machine learning based meta-layer in software-defined networks. In *2014 IEEE International Conference on Network Infrastructure and Digital Content*, pp. 121-125.

[34] Sendra, S., Rego, A., Lloret, J., Jimenez, J. M., & Romero, O. (2017). Including artificial intelligence in a routing protocol using software defined networks. In *2017 IEEE International Conference on Communications Workshops (ICC Workshops)*, pp. 670-674.

[35] Lin, S. C., Akyildiz, I. F., Wang, P., & Luo, M. (2016). QoS-aware adaptive routing in multi-layer hierarchical software defined networks: A reinforcement learning approach. In *2016 IEEE International Conference on Services Computing (SCC)*, pp. 25-33.

[36] Yu, C., Lan, J., Guo, Z., & Hu, Y. (2018). DROM: Optimizing the routing in software-defined networks with deep reinforcement learning. *IEEE Access*, *6*, 64533-64539.

[37] Glick, M., & Rastegarfar, H. (2017). Scheduling and control in hybrid data centers. In *2017 IEEE Photonics Society Summer Topical Meeting Series (SUM)*, pp. 115-116.

[38] Xu, J., Wang, J., Qi, Q., Sun, H., & He, B. (2018). Deep neural networks for application awareness in SDN-based network. In *2018 IEEE International Workshop on Machine Learning for Signal Processing (MLSP)*, pp. 1-6.

[39] Rezaei, S., & Liu, X. (2019). Multitask Learning for Network Traffic Classification. *arXiv preprint arXiv:1906.05248*.

[40] Lashkari, A. H., Draper-Gil, G., Mamun, M. S. I., & Ghorbani, A. A. (2017). Characterization of Tor Traffic using Time based Features. In *ICISSP*, pp. 253-262.

[41] ISCXFlowMeter (2016). Information security center of excellence, university New Brunswick. Retrieved from http://www.unb.ca/research/iscx/dataset /iscxflowmeter.html

[42] Mwadulo, M. W. (2016). A review on feature selection methods for classification tasks. *International Journal of Computer Applications Technology and Research*, *5*(6), 395-402.

[43] Chandrashekar, G., & Sahin, F. (2014). A survey on feature selection methods. *Computers & Electrical Engineering*, *40*(1), 16-28.

[44] Saeys, Y., Inza, I., & Larrañaga, P. (2007). A review of feature selection techniques in bioinformatics. *Bioinformatics*, *23*(19), 2507-2517.

[45] Lundberg, S. M., & Lee, S. I. (2017). A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems*, pp. 4765-4774.

[46] *F1 score*. (n.d.). Retrieved from https://en.wikipedia.org/wiki/F1_score.

[47] *Precision and Recall*. (n.d.). Retrieved from https://en.wikipedia.org/wiki/Precision_and_recall.

[48] Suárez-Varela, J., Carol-Bosch, S., Rusek, K., Almasan, P., Arias, M., Barlet-Ros, P., & Cabellos-Aparicio, A. (2019, August). Challenging the generalization capabilities of Graph Neural Networks for network modeling. In *Proceedings of the ACM SIGCOMM 2019 Conference Posters and Demos*, pp. 114-115.

[49] *Deep Learning* (n.d.). Retrieved from https://en.wikipedia.org/wiki/Deep_learning

[50] https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c.

[51] *Pycharm* (n.d.). Retrieved from https://www.jetbrains.com/pycharm/

[52] Qin, M., Lei, K., Bai, B., & Zhang, G. (2019). Towards a Profiling View for Unsupervised Traffic Classification by Exploring the Statistic Features and Link Patterns. In *Proceedings of the 2019 ACM Workshop on Network Meets AI & ML*, pp. 50-56.

[53] Artificial neural *network*. (n.d.). Retrieved from https://en.wikipedia.org/wiki/Artificial_neural_network.

[54] Salzberg, S. L. (1994). C4. 5: Programs for machine learning by J. Ross Quinlan. Morgan Kaufmann Publishers, Inc.

[55] *Random Forest Classifier* (n.d.). Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

[56] *Keras*. (n.d.). Retrieved from https://keras.io/