EAI Endorsed Transactions
on Scalable Information Systems
Research Article EAI.EU

# Prediction Based Dynamic Controller Placement in SDN

Ramya G[1,*], Manoharan R[1]

[1]Pondicherry Engineering College, Pondicherry, India

## Abstract

The current technologies such as IoT, 5G networks and Fog computing creates a challenge in the efficient management of devices in dynamic conditions. Software-Defined Network (SDN) has been defined as a promising solution for providing efficient network management by decoupling the data and control planes from the network devices and enables programmability of network devices. The major challenge in SDN is identification of number of controllers to be placed and its optimal placements in the network. To address this issue, this work proposes a Traffic Engineering mechanism that leverages the performance of Machine Learning to predict controller numbers by analysing and predicting the controller's traffic. The optimal locations of controllers are identified by using the K-Means++ algorithm. The proposed method is simulated using Mininet and the results depict that the proposed methodology outperforms the existing methodologies in terms of performance parameters.

**Keywords:** SDN, Quality of Controller, Traffic classification, Traffic Prediction, Controller Placement, Mininet, Flow Installation Time

Received on 13 December 2020, accepted on 18 April 2021, published on 27 April 2021



doi: 10.4108/eai.27-4-2021.169420

## 1. Introduction

The tremendous increase in the number of devices connected to the Internet and the current technologies like IoT, Fog, 5G needs an efficient network management system. The lack of efficient network management may result in network congestion, poor routing, and an increase in communication delay. The SDN and Network Function Virtualization (NFV) are key-enabling technologies to provide efficient network management. Both advocate open-source and programmability of network devices which leads to innovation in network management.

The capability of a network to deliver services rapidly by efficiently managing user demands is the aim of programmable networks. SDN aims to deliver a programmable network [1] by separating the control plane and data plane from the network devices and puts the control logic in the centralised control namely "the controller". The controller resides in the control plane of SDN knows the network topology created, number of forwarding devices and its current status, flow rule management and other network policies. This centralized nature of SDN allows us to enforce our traffic polices in the network [2]. Since the Controller in the network controls all the activities of the network, the controller's performance may affect the overall network performance. Therefore, the Controller should be highly efficient to manage the incoming PACKET_IN messages. It should not be under-utilized or should not be overloaded. Therefore, identifying the optimal number of controllers to be placed becomes essential.

This work adapts a Traffic Engineering Mechanism (TE) to identify the optimal number of controllers to be placed in the network. TE is an important application in network which aims to study the measurement and the management of network traffic [3]. Traffic Engineering is an iterative process of reading the network state, analysing and classifying the traffic and predicts future traffic to avoid network congestion. Traffic Engineering also focuses on enhancing the network performance in terms of Quality of Service, proper resource management, enhanced routing, improved security, and good user experience which may reduce cost also.

The openness of SDN by being programmable network allows user to do innovations and implementation of the

*Corresponding author. Email: ramya028@gmail.com

polices in the network and thus opens the doors for the implementation of new TE mechanisms. Several traditional TE mechanisms are available such as Equal Cost Multi-Path routing (ECMP), Intermediate System and Intermediate System (IS-IS), and Multi-protocol Label Switching (MPLS) [3] make all the decisions locally which may not be applicable in SDN because the controller in the SDN based network is centralized one.

Recently, Machine Learning (ML) is remarkably applied to every possible field including networking to solve complex issues. Applying ML techniques to the SDN network is easier because of the two major reasons. Firstly, the recent advancements in the Graphics Processing Unit (GPU) and the Tensor Processing Unit (TPU) [4] performs computations in a faster and efficient manner. Secondly, the SDN controller acts as a Network Operating System holds the global view of the entire network [2].

Though SDN has many advantages over traditional networks, two main issues namely Controller Placement Problem (CPP) and Flow Rule Management are to be addressed properly. This work mainly focuses on the CPP. The CPP can be defined as the identification of the number of controllers to be placed and their optimal locations in the network. This will not only enhance the performance of the network but also ensures the optimal utilization and fair load distribution amongst controllers. Therefore, the identification of the optimal number of controllers to be placed plays a vital role in improving the network performance. Once the controller numbers are identified, that has to be placed in the optimal location to avoid the unnecessary delay in the installation of flow rules to the switches. This work aims to identify the optimal number of controllers based on the network traffic. Here, the number of controllers is a variable that changes dynamically according to the network traffic. To achieve the above objectives, this work leverages the benefits of ML by combining with SDN to predict the controller numbers and their optimal placements in the network.

The placement of the controllers is done by K-Means++ clustering algorithm, which is an extended version of K-Means clustering technique. The K-Means++ performs the same way of K-Means however K-Means++ instructs the procedure to initialize the initial cluster positions. The initial number of controllers is identified and its placements are done by the K-Means++ algorithm. Once the placement is done, network traffic is generated, and the traffic are collected. From the collected traffic, the network traffic is analysed and predicted using the multi-class logistic regression model. The classification model is designed to predict LNH (Low, Normal and High) code. Based on the predicted traffic, the number of controllers for the LHN code are estimated exactly using a greedy approach and the numbers are sent to the placement algorithm for Optimal Placement of it.

The rest of the paper is organised as follows. Section 2 describes the Controller Placement Problem (CPP). In section 3 related works to CPP is discussed. Section 4 describes the proposed methodology for controller placement and for analysing and predicting network traffic. In section 5 a greedy approach is applied for distributing the load to controllers and how controller numbers are decided dynamically is presented. In section 6 results and comparisons are shown. Finally ends with a conclusion.

## 2. Controller Placement Problem (CPP)

In SDN networks, the Controller Placement Problem (CPP) can be defined as identifying the number of controllers to be placed in the network and its optimal locations for its placement. The Figure1 shows the SDN architecture. The Controller residing in the control plane is responsible for all network management activities. Whenever a packet is received by the switch, the switch looks into the flow table. If no match is found for that flow, the switch sends the packet as PACKET_IN message to the controller [1]. The Controller defines flow-rule for the packet and sends back to the switch as PACKET_OUT message. The Controllers are connected in a one-hop distance to the switch logically, but in reality, they are connected in multi-hop distance. The controller should be connected in such a way that the communication delay between the switch and the controller is minimal.

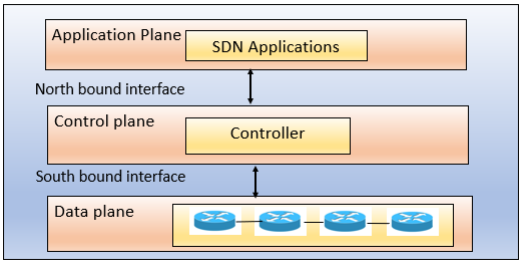


**Figure 1.** SDN architecture

During the early stage of SDN, a single controller [5] was deployed to control all the managing activities in the network. But "a single controller is always prone to 'single-point failure'". Hence the idea of placing multiple controllers in network was introduced [5]. The major setback with multiple controller scenario is deciding on the number of controllers have to be placed and where to place those controllers in the network. Figure 2 shows the controller connected in minimal delay with the switches.

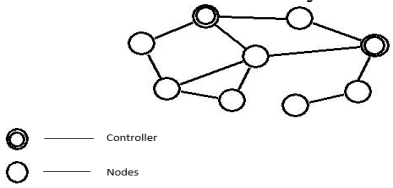


**Figure 2.** Controller- Switch connected with minimal latency

## 3. Related Work

The Controller Placement Problem (CPP) has attracted many researches to put forth their ideas for identifying optimal controller locations in the network. The major parameter considered for optimized controller placement was latency. The related work is categorised into two categories namely SDN Controller Placements and SDN Traffic Management as shown in the Figure 3. The first category presents the works related to CPP with "latency" as a parameter to be minimized and the second category presents the works related to SDN controller using machine learning methodologies.

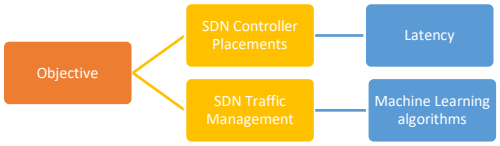


**Figure 3.** Related work on SDN Controller Management

### 3.1 Controller Placement based on Latency as objective parameter:

This section presents the related work in the CPP with respect to minimizing the latency between the controller and the switch. In [4], the CPP was addressed for the first time. The problem was mapped onto the Facility Location Problem to identify the optimal controller locations. The latency between the controller and the switches was taken as the parameter to optimize placements of controller.
In [5,6], the authors proposed a Pareto Optimal Controller Placement (POCO) and has taken more cases with respect to the latency parameter. In addition to that, the authors also considered link/node failure scenarios to elevate their work. However, the number of controllers to be placed was not calculated and was given directly to the algorithm for optimal placements of controllers. In [7], the authors proposed a linear programming model to minimize the Flow Setup Time by minimizing the distance between the switch and the controller in a distributed environment. In [8], the authors proposed a Firefly algorithm to minimize the communication latency between the controller and the switch.

In [9], a modified K-Means algorithm was proposed for the CPP in SDN. In that, for initial cluster selection, shortest path was calculated and the nodes with minimum distance were selected as the initial controller placements. Then using those points, the improved K-Means algorithm is used for optimal controller placement. In [10], a distributed controller layout problem was designed for the wide-area networks which aims to minimize the controller-to-controller latency and tried to balance the controller load. In [11], the authors proposed a Pareto Integrated Tabu Search (PITS) for optimal controller placements and also addressed the link/node failure and a greedy technique to perform migration in case of failure/controller load imbalance situations.

### 3.2 SDN Traffic Management:

This section gives insights to the work related to the machine learning algorithms proposed for the traffic management in SDN Controller. The Traffic management in SDN controller focussed more on classification of input traffic, providing Quality of Service (QoS) and identifying the network attack patterns. The traffic management in SDN is shown in Figure 4.

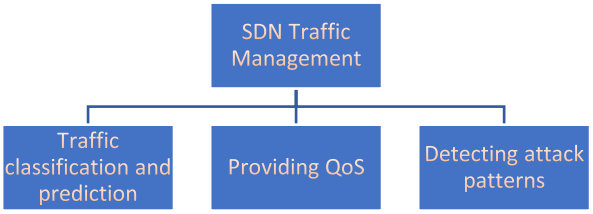


**Figure 4.** SDN Traffic Management

In [12], authors proposed a Decision Tree (DT) model to classify the input packet flow. This work classifies Elephant flows and Mice flows from the input packet and to predict the same. In [13], various machine learning algorithms both supervised and unsupervised algorithms were analysed to classify the traffic from various applications such as Facebook, YouTube, LinkedIn and other applications. The data for traffic classification were collected from the OpenFlow protocol of SDN and machine learning algorithms are applied to classify the traffic. Finally, they concluded that the Supervised Machine Learning techniques performs better than the Unsupervised methods. In [14], a QoS aware traffic classification method was achieved by using Semi-Supervised model and DPI. Initially, DPI was used to inspect the packets and then they were labelled. The labelled dataset was sent to the Semi-Supervised model for classification of applications and QoS was achieved. In [15], authors extended the SDN framework to the wireless network. Using the ML approach, the traffic classification was performed for mobile applications. They achieved an average of 95.5% classified data. By analysing the traffic, a framework was proposed and it was analysed for network performance using network parameters such as throughput, network lifetime.

In [16], an application awareness-based traffic classification was done. They created a framework called "Atlas" for the classification of various applications in android environment by incorporating ML and crowd sensing algorithm. In [17], they developed a Back Propagation Artificial Neural Network to analyse the

integrated load for various paths in the network and the same was predicted. Based on the prediction, routing optimization was achieved. In NeuRoute [18], introduced a dynamic routing by classifying traffic and predicts the traffic matrix from which dynamic routing was performed. In [19], a machine learning approach for controller placement was done. Multilabel traffic classification was done to predict the controller placement. The algorithm predicted whether the placed controller will be available in the future or not. A neural network-based algorithm is designed for predicting controller placement. In [20,21], Deep Neural Network was developed for predicting intrusion detection in the network.

In [22], the authors classified the controllers' traffic into Low and High based on the number of packets and number of bytes received by the controller using Artificial Neural Network (ANN) and predicted the controller traffic. In [23], the authors proposed a methodology to improve the quality of controller by connecting switches to more than one controller in 70-30 ratio. They improved the performance of controller in reliability context. Buts not focussed to optimize the number of controllers to be placed. In [28], the authors addressed the Link Flooding Attacks (LFA) in wired and wireless SDN by identifying its attack types and variants. They categorised and compared the existing mitigation on LFA on SDN ecosystems. In [29], the authors classified the network traffic based on the applications in the SDN network. They used three different supervised model namely Support Vector Machine, Naïve Bayes, and Nearest centroid for classification and also addressed the challenges faced during live data capturing.

From the existing work, it is obvious that either a mathematical model was formulated or a metaheuristic approach is mapped to solve CPP. When an ML approach is employed in SDN, either traffic classification or traffic prediction was performed. Classification primarily focussed on classification of application whereas prediction is done for network security purposes. Very few works have focussed on controller placement based on traffic classification and prediction. This work gives an insight in dynamic prediction of controller numbers and its optimal placement by classifying and predicting the network traffic using the Machine Learning technique.

## 4. Proposed Methodology

In this section, the proposed approach for controller placement and traffic prediction is described. The controller placement is done by K-Means ++ algorithm and the prediction of LNH code is done by using Multi-class Logistic Regression technique. The proposed approach is depicted in Figure 5.

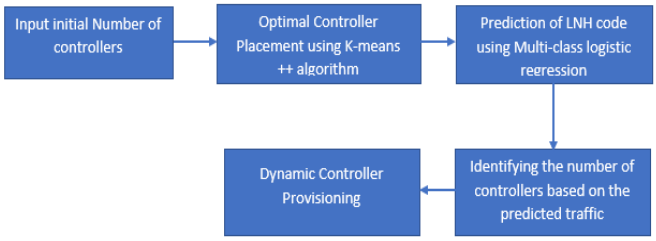


**Figure 5.** Proposed methodology

Initially, the controller positions are identified by means of the K-Means ++ initialization policy. Then the controllers are placed in the optimal locations using the same K-Means++ clustering technique. Once the controllers are placed in the optimal locations, the traffic is generated and captured. The captured traffic is classified as LNH code and the same is predicted. A greedy approach is proposed to distribute the predicted traffic among controllers and thus the number of controllers are identified and the numbers are given as input for the placement algorithm for its optimal placements. Here, we introduce a data manager to collect the traffic from the OpenFlow protocol, pre-processing it and sends the collected data to the Multi-class Logistic Regression algorithm for classifying and predicting the traffic.

### 4.1 Proposed Methodology for CPP

In the controller placement scenario, the communication latency between the controller and the switch should be minimum. Because, whenever a new packet arrives at the switch, the switch forwards the packet to the controller. The controller will decide on whether to forward/drop the packets. The controller processes the packets and sends forwarding rule to switches and it is shown in Figure 6.

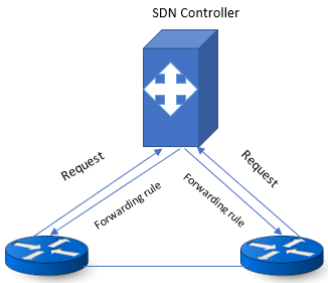


**Figure 6.** Packet Processing

So, the controller has to be placed in the network in such a way that the latency between the controller and switch is always minimum. This minimum latency reduces the packet flow installation time which eventually reduces the

average delay of the network [24]. If the controller is placed far away to the switch, then the time for installing the forwarding rule will get increased which eventually affects the network performance. Therefore, the distance-based K-Means++ clustering algorithm is adapted according to the CPP and optimal controller locations are identified.

The Controller Placement Problem can also be viewed as network partition problem. The clustering methodology is one of the methods to solve the network partition problem. The K-Means is the straight forward algorithm to solve the above issue. The major disadvantage in the K-Means is, it initially assumes the centroids randomly and tries to forms the clusters. In order to avoid it, the K-Means ++ algorithm was introduced. The K-Means ++ defines the procedure to initialize the positions of centroids in the network and adapts K-Means to form clusters. This works applies the same procedure to find the optimal locations to place controllers in the network.

In a network topology, G= (V, E) in which V represents the switches, E represents the link between the switches. Let C be the controllers to be placed in the network. The switches and the controllers are represented as follows:

V= {v1, v2, v3,…,vn}, where n denotes the number of switches in the network

C= {c1,c2,c3, …,ck}, where k denotes the number of controllers located in the network

The relation between the switch and controller can be defined as follows:

$$C \subset V$$

Let $X_v = \{(x_{v1}, y_{v1}), (x_{v2}, y_{v2}), \dots, (x_{vn}, y_{vn})\}$ be the position of switches in the topology and $X_c = \{(x_{c1}, y_{c1}), (x_{c2}, y_{c2}), \dots, (x_{ck}, y_{ck})\}$ be the position of controllers in the topology.

After placing the controllers in the network, the topology G can be defined as $G = (V, E, C)$. The first controller position is initialized uniformly random position in the network. This is similar to the K-Means algorithm but the key difference is that the other positions are not selected in the random manner. The D(x) is the distance parameter defines the distance between the initial controller position to the switches in the network. In the next step, the distance between the chosen controller position to all the switches in the network. The next controller position will be the one whose squared distance $(D(x))^2$ is the farthest to the initial controller position. Then from the second controller position, the next position will be calculated. The procedure gets repeated until the required number of controller positions are initialized.

1. Now with the initialized controller position, the K-Means algorithm is executed to get the optimal controller position. Specify the number of controllers to be placed in the network (k).
2. Initialize the initial controller positions in the network and assign switches to it.
3. For all switches connected to the controller, calculate the latency between the controller and switch by computing the sum of squared distance between the initialized controller position and the switches connected to it. It can be calculated as follows:

$$P = \sum_{i=1}^{n} \sum_{j=1}^{k} \left\| X_{vi} - X_{cj} \right\|^2 \text{ ------ (1)}$$

where, $\left\| X_{vi} - X_{cj} \right\|^2$ – squared distance between the switch and the controller.

4. If the initialized controller position is not optimal i.e. the latency between the controller and the switch is not minimum, then calculate the new controller position by using the following equation.

$$P_{nc} = \frac{1}{p} \sum_{i=1}^{n} X_{vi} \text{ ------- (2)}$$

where, $P_{nc}$ is the new controller position and $X_{cj}$ is the controller, which position to be recalculated.

5. Repeat Step 3 and 4 until no change in the controller positions.
6. Repeat the step 3,4, and 5 for other controllers to be placed in the network.

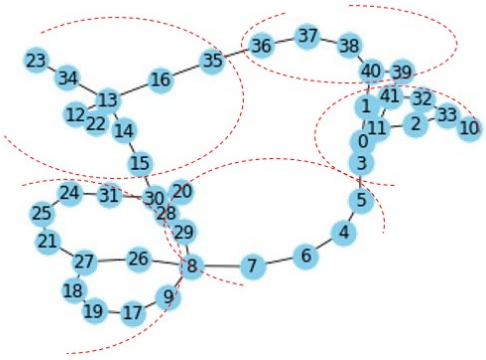


**Figure 7.** K-Means++ Clustering

The clusters formed for the sample network is shown in Figure 7 and the arcs represents the clusters formed. The workflow of the controller placement is shown in Fig 8.

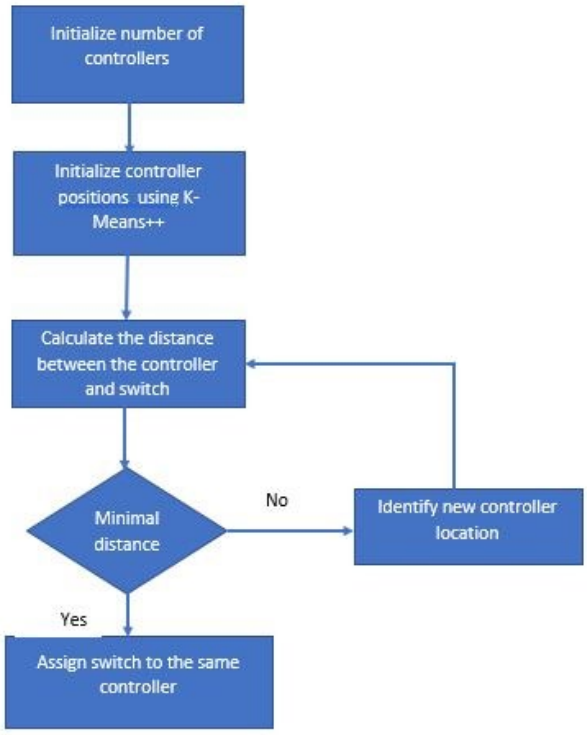


**Figure 8.** Work flow of Controller Placement technique

**Algorithm for Controller Placement:**

Input: G= (V, E), k, P (Placement set), $C = \{c_1, c_2, \dots, c_k\}$

Output: Optimal Controller location (P)

1. Select the initial position c from C
2. for all switches i in V:

$D(x) = \sum_{i=1}^{n} \|X_{c1} - X_{vi}\|$

3. Choose the next position of controller $c_i$, choosing $v \in V$ with the probability $\frac{D(x)^2}{\sum_{v \in V} D(x)^2}$
4. Repeat the steps until we have taken k initial controller positions

**Controller Placement:**

for all controllers (j) in k:
  for all switches $i \in j$ in n:
    **min:** $P_j = \sum_{i=1}^{n} \|X_{vi} - X_{cj}\|^2$
    if $P_j$ is minimal
      $P \leftarrow P_j$
    else
      $P_j' = \frac{1}{P_j} \sum_{i=1}^{n} X_{vi}$
      goto min
  j++
return P

The controller positions for the topology LambdaNet is shown in Figure 9. Initially, 5 controllers are placed in the network. So, the number of clusters was initiated to 5. By calculating the latency between the centers (controllers) and the switches, the nodes which are assigned as controllers for LambdaNet network shown in Figure 7 are [1,40], [7,8], [13,34], [21,27], [36,37]. The controller and nodes assigned as follows:

Cluster A: [40,2,33,41]
Cluster B: [0,3,2,11,4,5,6,8,9,12,13,16,14,15,17]
Cluster C: [8,26,29,15,24,30,35]
Cluster D: [17,19,18,27,20,30,25]
Cluster E: [32,35,36,37,40]

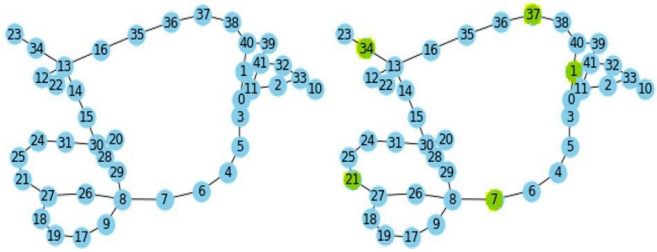

**Figure 9.** LambdaNet network and its controller placements

## 4.2 Traffic Classification and Prediction (TCP) SDN:

In this section the Traffic Classification and Prediction (TCP SDN) is discussed. Traffic Classification and Prediction plays a vital role in network traffic management. The network resources can be efficiently utilized by managing the network traffic. This work, addresses the proper utilization of the controllers placed in the network by classifying and predicting the network traffic. In this work, the traffic is classified as Low, Normal, and High (LNH code). By predicting the traffic, the number of controllers to be placed can be varied dynamically by adding more controllers or removing some controllers in the network. Here, the Multi-Class Logistic Regression [25] is used to predict LNH code.

The entire work flow of this work is shown in Figure 10. The proposed TCP SDN consists of the Traffic Monitoring (TM) module, Data Preparation (DP) module, and Traffic Analysis and Prediction modules (TAP).

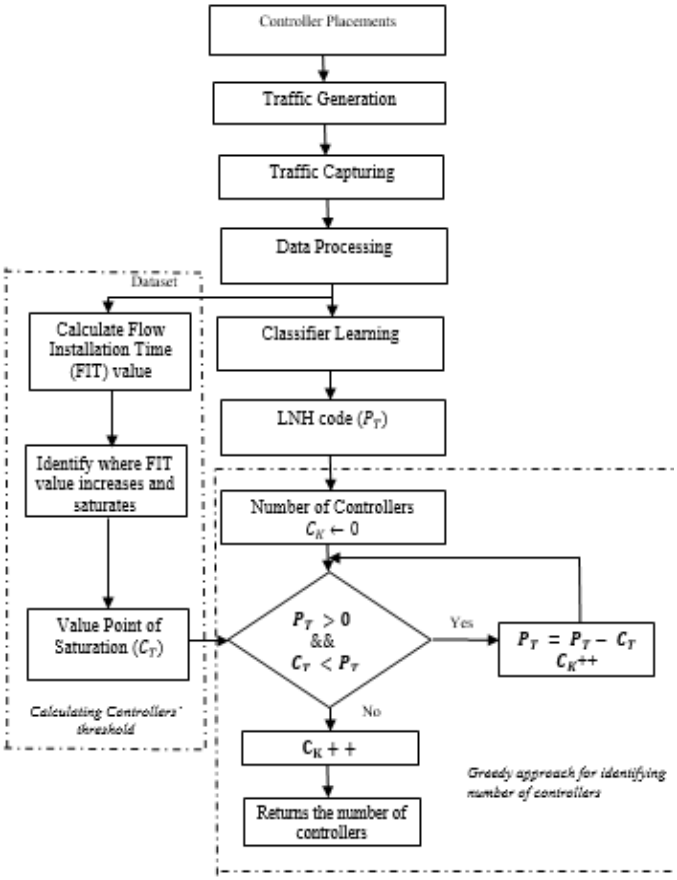


**Figure 10.** Work flow of the Proposed Prediction mechanism

The above modules will perform the following operations:

1. Traffic generation
2. Traffic capturing
3. Data Pre-Processing and Feature Extraction
4. Model learning
5. Classification and
6. Prediction

**Traffic Monitoring Module:**

This module performs Traffic generation and capturing. The network traffic is generated by using Iperf commands. The network traffic statistics can be collected from the OpenFlow Switches and from the Wireshark tool. The data is collected on that day and its previous day and combined together to form the dataset for data pre-processing.

**Data Preparation Module:**

The traffic management mechanism proposed in this paper, performs analysing the data, prediction of data, and calculating the threshold value for classifying LNH code. To perform this, the dataset needs to be transformed to the required format to train the model. The data manager resides here and performs the data transformation. The data manager starts data cleaning process starts by removing the unwanted attributes from the dataset.

The first field in the dataset is time and is made atomic and the corresponding value are mapped to identify how much traffic is been generated at that time. The dataset consists of different types of data format which needs to be normalized (0-1) to the single format to make it suitable for the training. The standard scalar function is utilized to normalize the data. The standard value of a data point can be calculated as follows:

$$z = \frac{(x-u)}{s} \quad \text{--- (3)}$$

where, z is the normalized score, x is the value to be normalized, u is the mean and s is the standard deviation value. The attribute Controller is added to the dataset which is computed by identifying the switch to which controller it is connected. The final attribute Traffic is also added by analysing the attributes Time, N_Packets (number of packets transferred at that time) and N_Bytes (number of bytes transferred at that time). An average mean value is computed for the N_Packets and N_Bytes at the time T is used to set the threshold value for classifying the LNH code. The final dataset which is to be sent to the classifier algorithm contains 9 attributes in which 8 are independent attributes and 1, the target attribute is the dependent one. The data manager performs all the data cleaning and pre-processing process and make the dataset ready for analysis.

**Traffic Analysis and Prediction (TAP):**

The TAP module performs classifier learning and predicting the LNH code. The classifier learning is the most important task in any machine learning process. The dataset is reduced to 70% with all the 9 attributes is set to learning phase. The remaining 30% of the dataset is set for testing. The dataset obtained contains 135678 data values on which 70% (94975) is set for training the classifier. The remaining 30% (40702) data values are set for testing.

The classifier learning process begins deciding the solver, deciding C value (Inverse regularization strength), deciding multi- class value and tuning the hyperparameter $\beta$ by computing its value. After deciding these values, the input data to the classifier learning is the Feature Matrix (FM). The FM consists of all the attributes in which analysis operation going to be performed.

Suppose, the dataset has 'm' features and 'n' observed values, the Feature Matrix (FM) is represented as follows:

$$FM = \begin{pmatrix} 1 & a_{11} & \dots a_{1m} \\ 1 & a & \dots a_{2m} \\ \vdots & \vdots & \vdots \ \vdots \\ 1 & a_{nm} & \dots a_{nm} \end{pmatrix}$$

The number of classes needs to be classified are Low, Normal, and High (LNH code) and thus number of classes becomes 3 (k=3). In this proposed approach, the solver is set to 'lbfgs' and the multi-class is set to 'OVR'. The OVR (One-vs-Rest) approach consider each class as binary model and applies regression on it.

Let $Y_i$ be the outcome corresponding to the 'k' classified class. The predictor function $f(k,i)$ is to be predicted for the $i^{th}$ row can be calculated as follows:

$$f(k,i) = \beta_{0,k} + \beta_{1,k}\, a_{1,i} + \beta_{2,k}\, a_{2,i} + \cdots + \beta_{M,k} a_{M,i} \text{ ---- (4)}$$

where, $\beta$ is the regression coefficient to be tuned.
The Logistic Regression is a predictive analysis algorithm that is based on the probability of the number of events occurred. The probabilities for LNH code are as follows:
$P(Y_L = 1 \vee X; \theta)$ for class 1 (low traffic)
$P(Y_N = 2 \vee X; \theta)$ for class 2 (normal traffic)
$P(Y_H = 3 \vee X; \theta)$ for class 3 (high traffic)
To fit the model,

$h_\theta{}^{(L)} = P(y = i \vee X; \theta)$ for low traffic, $h_\theta{}^{(N)} = P(y = i \vee X; \theta)$ for Normal traffic, and

$h_\theta{}^{(H)} = P(y = i \vee X; \theta)$ for high traffic.

The regression model for TCP SDN is shown in Figure 11.

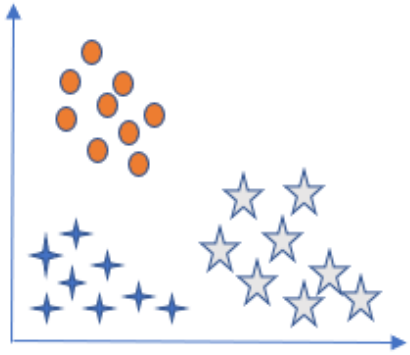

**Figure 11.** Traffic classification

Here, ● represents low traffic, ✦ represents medium/normal traffic and high traffic is represented by ☆. The model for LNH code is represented in Figure 12.

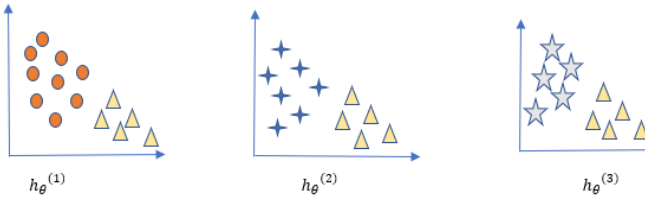


**Figure 12.** LNH code classification

The last layer of Multi- Class logistic Regression is SoftMax regression which integrates all the probability values to produce the final predicted traffic ($P_T$) . Given the input to the SoftMax layer, it performs the exponential operation to all the input values (z) and makes all the values to be positive. The SoftMax function is defined as follows:

$$softmax(z) = \frac{e^z}{\sum_{i=1}^{n\_classes} e^{z_i}} \quad \text{---- (5)}$$

The workflow of classifying and prediction is shown in Figure 13.

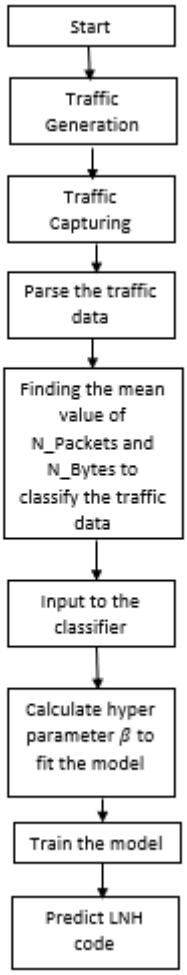


**Figure 13.** Work Flow of TAP module

**Algorithm for Traffic Prediction:**

Input: Network Traffic Dataset
Output: LNH code

1. Generate Traffic using Traffic generating commands.
2. Obtain Flow Table Statistics from OpenFlow Switches and from the Wireshark tool.
3. The data cleaning process is carried out to transform the attributes and normalize the data to uniform format.
4. Feature Extraction is carried out and attributes such as Controller and Traffic are calculated.
5. Derive Feature Matrix (FM) from the selected attributes.
6. Split Train (70%) and Test (30%) dataset from the traffic dataset and tune the hyper parameter $\beta$.
7. Perform the Classifier learning and predict LNH code.

### 4.3. Decision of Number of Controllers using Greedy approach:

From the obtained traffic dataset, Controller Threshold ($C_T$) is set by analysing the network traffic. To find out the $C_T$, the Flow Installation Time (FIT) value is calculated. The point at which the FIT value increases and saturates that point is taken as the $C_T$. The number of controllers needed for the network is decided by using both the Predicted Traffic ($P_T$) and the Controllers' Threshold $C_T$. Each and every controller to be placed in the network is considered as a bin. The Predicted Traffic is allotted to the controller up to its threshold value $C_T$. The Predicted Traffic gets subtracted after the traffic is allotted to the controller. The number of controllers in which the Predicted Traffic is allotted is taken as the number of controllers needed for the network. By identifying and optimizing the number of controllers to be placed in the network, the controller quality can be improved and eventually network performance also increases.

**Algorithm for Deciding the Number of Controllers:**

Input: Predicted Traffic ($P_T$), Controller threshold ($C_T$)

Output: Number of Controllers

1. Initialize number of controllers ($C_K$) to 0
2. While ($P_T$>0)
3. If ($P_T > 0$ && $C_T < P_T$)
   - $P_T = P_T - C_T$
   - $C_K++$
   - Continue;
   - Else
     - $C_K++$
     - $P_T = 0$
4. Return $C_K$

## 5. Result and Analysis

### 5.1. Experimental Setup and Performance Metrics for Controller Placement

This section presents the experimental setup and the performance metrics taken to examine the performance of the proposed method. The proposed K-Means++ algorithm is written in Python and the controller placement scenario is executed in the Mininet emulator. The Mininet Emulator is an exclusive environment to simulate Software-Defined Networks [26]. The controller used for the experiment is the POX controller and the OpenFlow switch version is OF 1.3. Many network topologies are considered for executing the K-Means++ algorithm. The topologies are taken from the standard Internet Topology Zoo [27] and the topologies considered are LambdaNet, IRIS, Forthnet, BTN, Bellsouth, Arpanet, Abvt, and Sprint.

The proposed K-Means++ is evaluated by comparing the proposed with the existing algorithm Pareto Integrated Tabu Search (PITS) [5], Genetic Algorithm (GA), standard K-Means, and with the Random Placements (RP). All the algorithms are executed numerous times and the best of it is shown here. In all the runs, the proposed approach outperforms the other two approaches. Additionally, the following situations are implemented to evaluate the controller performance.

A single controller scenario. A single controller is placed in the optimal location of the network and the performance was analysed.

The controller and the switches are far away from each other. Here, the switches are purposefully connected to the controller which is far away to the switches. So that the performance of the controller can be evaluated.

In the SDN scenario, the controller plays a vital role in evaluating network performance. So, the time taken to install the flows to the switch is taken as the major parameter for evaluating the controller performance. The other parameters taken are Average delay and Network Throughput.

**Flow Installation Time (FIT):**

Flow Installation Time is, time taken by controller to analyse packet and sending forwarding rules to switches. It can be calculated as follows:

$$L_{fit} = \left((L_{sc} * numberofPACKET_{IN}) + (L_{cs} * numberofPacket_{OUT})\right) \text{ --- (4)}$$

where,

$L_{SC}$ – Latency between the switch and the controller

$L_{CS}$ – Latency between the controller and switch

$L_{CS}$ can be calculated as follows:

$$L_{CS} = \sum_{i=0}^{|V|} \sum_{j=0}^{k} d(S_i, C_j) \quad \text{--- (5)}$$

where,

$d(S_i, C_j)$ defines the latency between the switch and the controller

k defines number of controllers

**Average Delay:**

The Average Delay is defined as the delay recorded for installation of flow rules and the time to transmit the packet.

The Average Latency ($D_{AVG}$) can be calculated as follows:

$$D_{Avg} = \frac{P_{AT}+L_{fit}+P_{FT}}{numberofpackets} \text{ ---- (6)}$$

where,

$P_{AT}$ defines Packet Arrival Time

$L_{fit}$ defines Flow Installation Time

$P_{FT}$ defines Forwarding Time

**Throughput:**

It is defined as the average data rate of successful transmission of data. It is defined in bits/s.

### 5.2 Results and Comparison of CPP:

**Flow Installation Time:**

The FIT for all the algorithms is calculated and presented in Figure 14. The FIT value of the proposed K-Means++ shows an improvement in lowering the time for installing flow rules to switches. The proposed is compared with the existing algorithms and the proposed algorithm performs on an average of 60-70 % (approximately) better than the other methodologies.

The lowest recorded FIT value is 10ms in LambdaNet topology for the proposed K-Means++ algorithm and the high FIT value recorded for the proposed approach is 50ms. The average FIT value of the proposed approach is 32ms approximately which is quite low when compared with the other mechanisms taken for comparison. From the graph, it is evident that the proposed K-Means++ outperforms the other algorithms.

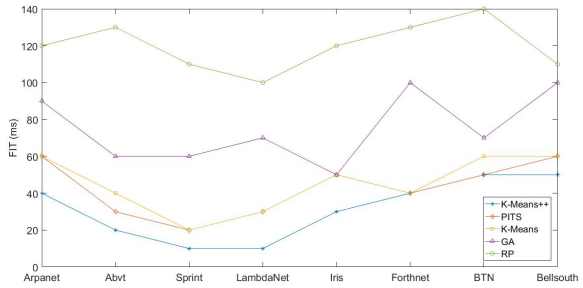


**Figure 14.** Flow Installation Time

**Average Latency:**
The Figure 15 illustrates the recorded average delay for transmitting packets from the source to destination of all the algorithms in different topologies.

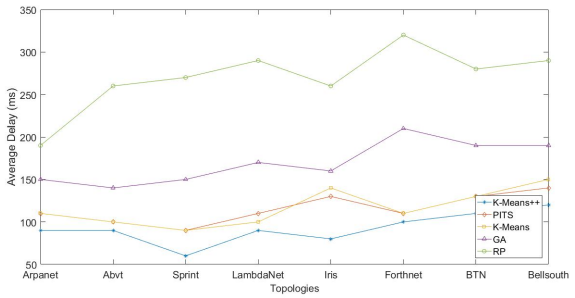


**Figure 15.** Average Latency

The average delay is recorded very low for the proposed approach which is less than 50ms. The average latency value of proposed shows improvement in lessening the average delay of network for all the topologies. When the proposed work is compared with other methodologies, the average delay decreased to 55-67% approximately which certainly shows the greater performance of the proposed approach. The average value of average delay of the proposed approach is very low when compared with other algorithms.

**Throughput:**
The Figure 16 depicts the throughput obtained for various topologies of the proposed and the other algorithms taken for comparison.

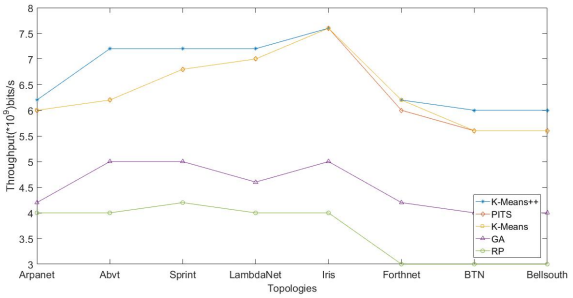


**Figure 16.** Throughput

The average throughput of the proposed methodology is 6.7*10^9 bits/s. The higher throughput gained for the proposed K-Means++ is in the IRIS topology. From the above graph, it is evident that the proposed performs better than the other methods considered here. On an average of 20-30 % improvement in throughput of proposed approach when compared with other methods.

## 5.3 Experimental Results for the Single Controller Scenario:

A single controller is placed in the location of the network using the proposed K-Means++ approach. A single controller is placed so that the performance of the network as well as the controller can be evaluated. When a single controller was placed in the network, the single controller is able to respond (PACKET_OUT) to all the incoming PACKET_IN messages received from all the switches presented in the network. The overhead occurred in this scenario is, the time to install the forwarding rules to the switches.

Since, a single controller was placed, it is its responsibility to process all the incoming packets and make decision to all the incoming PACKET_IN messages. The FIT got increased which ultimately affects the average delay. The switches which are closer to the controller got the forwarding rules installed quicker than the switches that are far away from the controller.

The topologies in which the single controller was placed are Iris, Forthnet, BTN, Chinanet, and LambdaNet. For the experimental purpose, a single controller was placed using K-Means++ algorithm and the metrics were analysed. For the comparison purpose, a single controller was placed randomly in the network and the performance was analysed. In both the methods of placing a single controller, K-Means++ performed much better than the random placements of controller in the network. The Figure 17,18,19 presents the FIT, Average delay and throughput of K-Means++ and Random Placement of single controller.

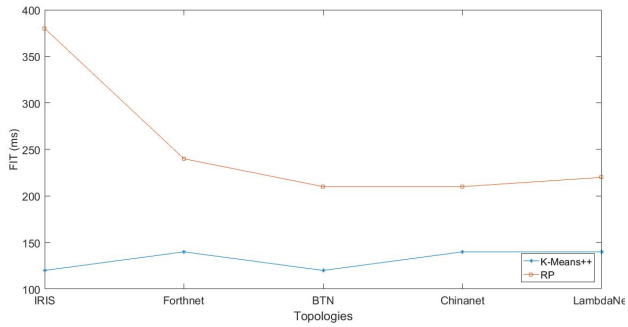


**Figure 17.** Flow Installation Time

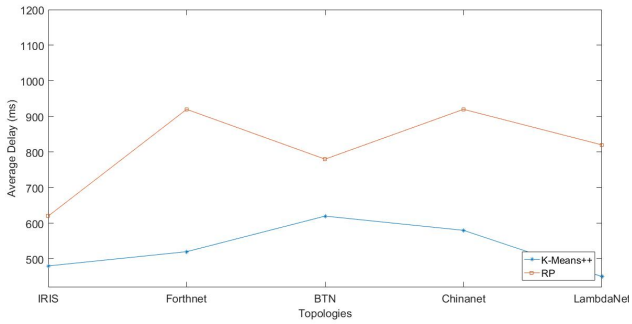


**Figure 18.** Average Latency

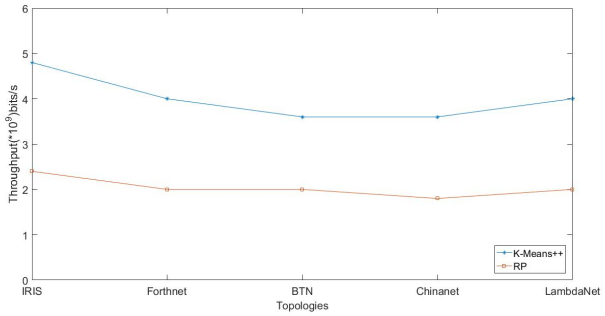


**Figure 19.** Throughput

## 5.4 Controller and Switches Increased Latency:

In this scenario, the controller and switches are placed in such a way that the controller and switches are far away from each other. In order to analyse the FIT for the PACKET_IN messages, this scenario is simulated. Here, a single controller is placed anywhere in the network. The same network commands used in above cases are used to generate traffic. In this scenario, particularly the switch which is connected far away from the controller and the switch which is nearer to the controller is identified and those switch performances were analysed for the topologies includes Iris, Forthnet, BTN, Chinanet, and LambdaNet. The Figure 20 shows the FIT when the switch is connected to the nearest controller and to the farthest controller.

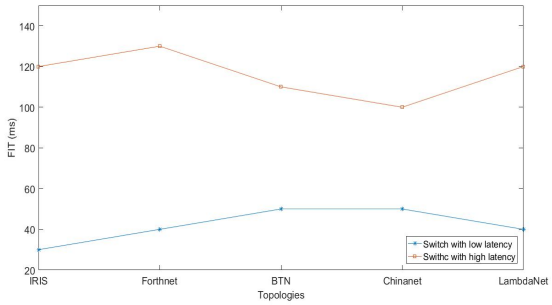


**Figure 20.** FIT when the switches are connected to the nearest and farthest controller

From the observed result, it is evident that the switch which is connected to the closest controller will have low FIT. So, the controller should be placed in the network in such a way that the latency between the controller and the switches connected to it is low.

## 5.5 Experimental setup and Performance metrics for Traffic Analysis and Prediction:

This section explains the experimental setup for traffic analysis and prediction. The Multi-class Logistic Regression algorithm is written in Python and executed using Juypter Notebook. The proposed model is compared with relevant multi-class models. The models taken for comparing our proposed work are Naïve-Bayes and Support Vector Machine.

**Performance Metrics:**

Precision, Recall, f1score, accuracy and hamming loss are used as performance metrics to evaluate the performance of the model.

**Precision:**

It is defined as number of true positive rate ($T_p$) over the sum of number of true positives and false positives ($F_p$) and it is calculated by using equation 7.

$$Precision = \frac{T_p}{T_p + F_p} \quad \text{---- (7)}$$

**Recall:**

It is defined as the number of true positives over the sum of true positives and the number of false negatives ($F_n$) and it is calculated as in equation 8.

$$Recall = \frac{T_p}{T_p + F_n} \text{---- (8)}$$

**F1-Score:**

The F1-score is defined as the "harmonic mean of Precision and Recall". This can be calculated by using equation 9.

$$\text{F1-Score} = 2 * \frac{(precision * recall)}{(precision + recall)} \quad \text{---- (9)}$$

The following are the precision, recall, and F1-score values obtained from the experiment.

| | precision | recall | f1-score |
|---|---|---|---|
| 1 | 1.00 | 1.00 | 1.00 |
| 5 | 1.00 | 1.00 | 1.00 |
| 10 | 0.97 | 1.00 | 0.98 |
| accuracy | | | 1.00 |
| macro avg | 0.99 | 1.00 | 0.99 |
| weighted avg | 1.00 | 1.00 | 1.00 |

From the precision – recall values shown above, it is evident that the multiclass logistic regression performs well in predicting the values. Here, 1 stand for low traffic, 5 stands for normal/regular traffic, and 10 stands for high traffic. The multi class LR is compared with Naïve Bayes, Support Vector Machine (SVM). The Figure 21 shows the comparison of various models.

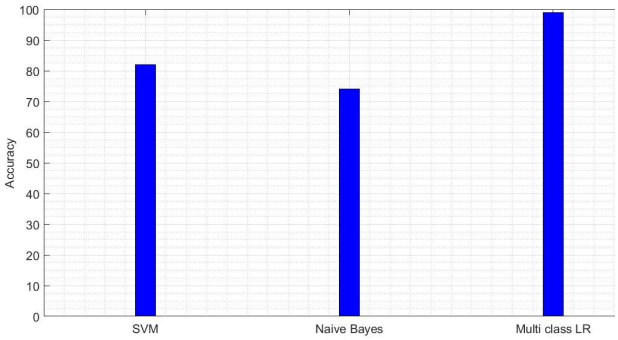


**Figure 21.** Model Accuracy

**Hamming Loss:**

It can be defined as the "fraction of incorrectly predicted samples" from the dataset. The following Figure 22 shows the hamming loss of each model.

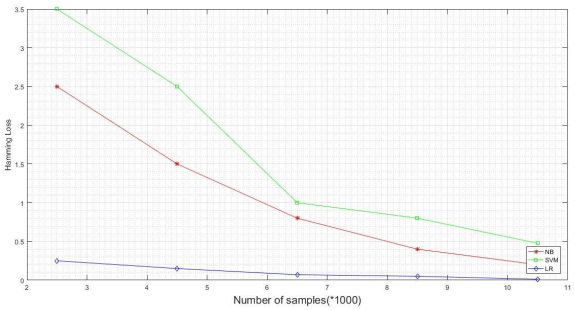


**Figure 22.** Hamming Loss

## 5.6 Number of Controllers after Traffic Prediction:

The future traffic of the network is predicted and from which the number of controllers needed for the predicted traffic is calculated. The number of controllers required for controlling the predicted network traffic for various topologies during low, regular, and high traffic are shown in Figure 23.

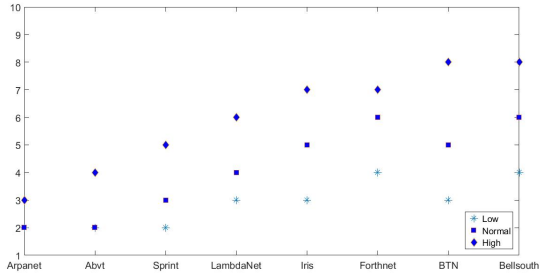


**Figure 23.** Predicted Number of Controllers

The predicted number of controllers sent as the input to the K-Means++ algorithms and are placed in optimal locations of the network. The controllers' number can be either increased or decreased according to the network traffic and this assures the dynamic decision on controllers' number to be placed in the network.

## 5.7 Effect of Varying the Controllers' Threshold:

Here, the performance of network is also analysed by varying the threshold value from 1000kreq/s to 2000kreq/s. The following graphs are the plots of controllers' performance with respect to varying threshold. The Figure 24 depicts the FIT value for various threshold.

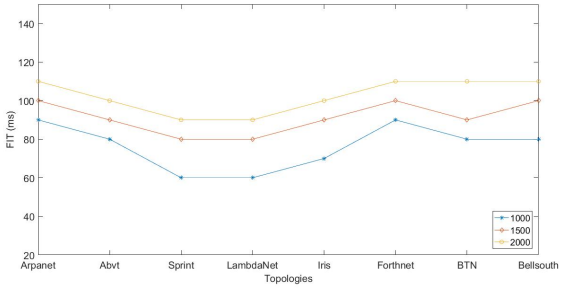


**Figure 24.** FIT

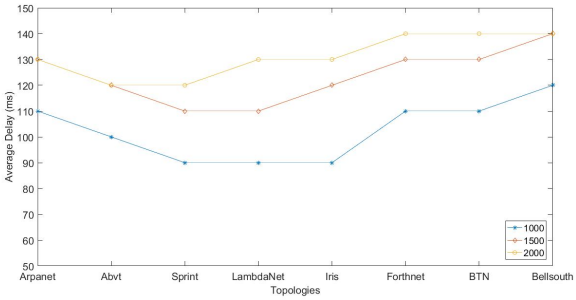


**Figure 25.** Average Latency

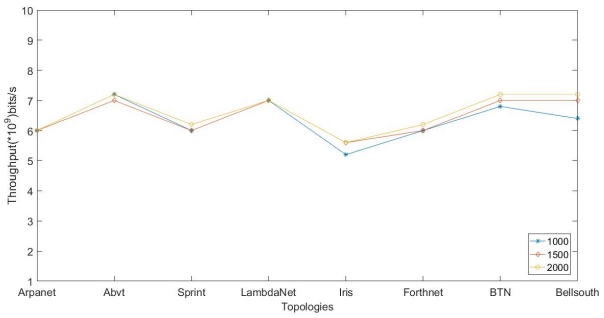


**Figure 26.** Throughput

The above Figure 25 and 26 shows the network performance with different controller threshold values in terms of Average Latency and Throughput of proposed K-Means++ for various topologies.

## 6. Conclusion

In SDN, the controller plays a vital role in network management. Since all the network activities are managed by the controller, it is essential to find optimal number of controllers and place them in optimal locations. In this work, we proposed a method which dynamically decide the number of controllers and their optimal placement in the network. The proposed model analyses and predict the network traffic and classify it into LNH code. This was achieved using the Multi-class Logistic Regression model. From the predicted traffic (LNH code), a greedy approach was designed to decide the optimal number of controllers to manage the predicted traffic. Then the optimal number of controllers are placed in the optimal locations in the network using the adapted K-Means++ algorithm. The propose methods were analysed for various performance and compared with the existing methodologies. From the experimental results, it is evident that the proposed approach performs better than the existing works. In the future, this work can be extended to detect network attack patterns.

## References

[1] Singh, Ashutosh Kumar, and Shashank Srivastava. *"A survey and classification of controller placement problem in SDN."* International Journal of Network Management 28, no. 3 (2018): e2018.

[2] Xie, Junfeng, F. Richard Yu, Tao Huang, Renchao Xie, Jiang Liu, Chenmeng Wang, and Yunjie Liu. "*A survey of machine learning techniques applied to software defined networking (SDN): Research issues and challenges*." IEEE Communications Surveys & Tutorials 21, no. 1 (2018): 393-430.

[3] Wang, Mowei, Yong Cui, Xin Wang, Shihan Xiao, and Junchen Jiang. "*Machine learning for networking: Workflow, advances and opportunities*." IEEE Network 32, no. 2 (2017): 92-99.

[4] Heller, Brandon, Rob Sherwood, and Nick McKeown. "*The controller placement problem*." In Proceedings of the first workshop on Hot topics in software defined networks, ACM,(2012), pp. 7-12.

[5] Lange, Stanislav, Steffen Gebert, Thomas Zinner, Phuoc Tran-Gia, David Hock, Michael Jarschel, and Marco Hoffmann. "*Heuristic approaches to the controller placement problem in large scale SDN networks*." IEEE Transactions on Network and Service Management 12, no. 1 (2015): 4-17.

[6] Hock, David, Matthias Hartmann, Steffen Gebert, Michael Jarschel, Thomas Zinner, and Phuoc Tran-Gia. "*Pareto-optimal resilient controller placement in SDN-based core networks.*" In 25th IEEE International Teletraffic Congress (ITC), IEEE (2013), pp. 1-9.

[7] Sridharan, Vignesh, Mohan Gurusamy, and Tram Truong-Huu. "*On multiple controller mapping in software defined networks with resilience constraints.*" IEEE Communications Letters 21, no. 8 (2017): 1763-1766.

[8] Sahoo, Kshira Sagar, Sampa Sahoo, Anamay Sarkar, Bibhudatta Sahoo, and Ratnakar Dash. "*On the placement of controllers for designing a wide area software defined network.*" In TENCON 2017-2017 IEEE Region 10 Conference, pp. 3123-3128. IEEE, 2017.

[9] Wang, Guodong, Yanxiao Zhao, Jun Huang, Qiang Duan, and Jun Li. "*A K-means-based network partition algorithm for controller placement in software defined network.*" In 2016 IEEE International Conference on Communications (ICC), pp. 1-6. IEEE, 2016.

[10] Liao, Lingxia, and Victor CM Leung. "*Genetic algorithms with particle swarm optimization-based mutation for distributed controller placement in SDNs.*" In 2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), pp. 1-6. IEEE, 2017.

[11] Ramya, G., and R. Manoharan. "*Enhanced Multi-Controller Placements in SDN.*" In 2018 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET), pp. 1-5. IEEE, 2018.

[12] Xiao, Peng, Wenyu Qu, Heng Qi, Yujie Xu, and Zhiyang Li. "*An efficient elephant flow detection with cost-sensitive in SDN.*" In 2015 1st International Conference

on Industrial Networks and Intelligent Systems (INISCom), pp. 24-28. IEEE, 2015.

[13] P. Amaral, J. Dinis, P. Pinto, L. Bernardo, J. Tavares, and H. S. Mamede, "*Machine learning in software defined networks: Data collection and traffic classification*," in Proc. IEEE ICNP'16, Singapore, Singapore, Nov. 2016, pp. 1–5.

[14] Wang, Pu, Shih-Chun Lin, and Min Luo. "*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. IEEE, 2016.

[15] Uddin, Mostafa, and Tamer Nadeem. "*TrafficVision: A case for pushing software defined networks to wireless edges*." In 2016 IEEE 13th International Conference on Mobile Ad Hoc and Sensor Systems (MASS), pp. 37-46. IEEE, 2016.

[16] Z. A. Qazi, J. Lee, T. Jin, G. Bellala, M. Arndt, and G. Noubir, "*Application-awareness in SDN*," in Proc. ACM SIGCOMM'13, Hong Kong, China, 2013, pp. 487–488.

[17] Chen-Xiao, Cui, and Xu Ya-Bin. "*Research on load balance method in SDN*." International Journal of Grid and Distributed Computing 9, no. 1 (2016): 25-36.

[18] Azzouni, Abdelhadi, Raouf Boutaba, and Guy Pujolle. "NeuRoute: Predictive dynamic routing for software-defined networks." In 2017 13th International Conference on Network and Service Management (CNSM), IEEE, 2017, pp. 1-6.

[19] S. Nanda, F. Zafari, C. DeCusatis, E. Wedaa, and B. Yang, "*Predicting network attack patterns in SDN using machine learning approach*," in Proc. IEEE NFV-SDN'16, Palo Alto, CA, USA, Nov. 2016, pp. 167–172.

[20] T. Tang, S. A. R. Zaidi, D. McLernon, L. Mhamdi, and M. Ghogho, "*Deep recurrent neural network for intrusion detection in SDN-based networks*," in Proc. IEEE NetSoft'18, Montreal, Canada, 2018.

[21] N. Shone, T. N. Ngoc, V. D. Phai, and Q. Shi, "*A deep learning approach to network intrusion detection*," IEEE Trans. Emerging Topics in Computational Intelligence, vol. 2, no. 1, pp. 41–50, Feb 2018.

[22] Thiruvengadam Hemamalini, Ramya Gopalakrishnan, and Manoharan Rajendiran. "*Dynamic Controller Deployment in SDN Networks Using ML Approach*." In International Conference on Sustainable Communication Networks and Application, pp. 311-318. Springer, Cham, 2019.

[23] Sridharan, Vignesh, Purnima Murali Mohan, and Mohan Gurusamy. "*QoC-Aware Control Traffic Engineering in Software Defined Networks*." IEEE Transactions on Network and Service Management (2019).

[24] G Ramya, R Manoharan, "*A New Algorithm for Controller Placement in SDN*", International Journal of Engineering and Technology (IJEAT), Vol.8, pp. 1196-1201, (2019)

[25] Bishop, Christopher M. (2006) "*Pattern recognition and machine learning*",springer.

[26] Kaur, Karamjeet, Japinder Singh, and Navtej Singh Ghumman. (2014) "*Mininet as software defined networking testing platform*." In International Conference on Communication, Computing & Systems (ICCCS), pp. 139-42.

[27] Knight, Simon, Hung X. Nguyen, Nickolas Falkner, Rhys Bowden, and Matthew Roughan.(2011) "*The Internet Topology Zoo*." IEEE Journal on Selected Areas in Communications 29, no. 9 1765-1775.

[28] ur Rasool, Raihan, Hua Wang, Usman Ashraf, Khandakar Ahmed, Zahid Anwar, and Wajid Rafique. "*A survey of link flooding attacks in software defined network ecosystems*." Journal of Network and Computer Applications (2020): 102803.

[29] Raikar, Meenaxi M., S. M. Meena, Mohammed Moin Mulla, Nagashree S. Shetti, and Meghana Karanandi. "*Data Traffic Classification in Software Defined Networks (SDN) using supervised-learning*." Procedia Computer Science 171 (2020): 2750-2759.