A QoS and Load Balancing Predictive Model Based on LSTM and Random Forest Regression in SDN: A Rest API Approach

Muhammad Salman Ansari[1(✉)], JianXun Zhang[1], and Shaban[2]

[1] Tianjin University of Technology and Education, Tianjin, China
4usalman9999@gmail.com, zhangjx@tute.edu.cn
[2] University of Electronic Science and Technology, Chengdu, China

Abstract. Fog Computing, Software Define Networking, RESTful API and Machine Learning are new technologies in the area of ICT as well as in Networking. Fog computing brought high performance computing at the edge of network and Software Defined networking. As we intended to head towards QoS and Load balancing in SDN. In this work we cover an initial segment which is Machine leaning model implementation on a network traffic information to predict future outcomes for traffic flow. Meanwhile, in comprehensive research we intended to reduce network congestion, jitter, QoS and load balancing by learning past traffic flow information. As well as, we aim to improve maximum utilization of link-bandwidth. Through RESTful API of SDN, we can embed Machine learning based prediction model to the network server which can modify the network policies by learning from past experiences. In this paper, we proposed a QoS and load balancing framework. A Server and SDN based application for network monitoring, management and controlling the policies over network gateway for better performance in regard of traffic flow. Experiment result shows that implementation of Machine learning over network traffic flow information immensely important for new emerging technologies. The evaluation results of Machine learning model we implemented in this work depicts that model performs well. Meanwhile the model improves by increasing with the number of epochs.

Keywords: Machine Learning in Networking · SDN · RESTful API · QoS and Load balancing

1 Introduction

Although, the world is transforming communication technologies to a distributed approach for efficient, faster, and more reliable services, which enables new challenges in handling multiple technologies on a single platform at the inception of wireless sensing IoT, Wireless LANs, and traditional LANs infrastructure, known as Heterogeneity, load balancing, etc. The emergence of new technologies brings about new structures and framework deployments alongside to meet demand, which is directly proportional to the increase in the cost of deployment in infrastructure, management, and performance monitoring.

Y. Cao and X. Shao (Eds.): DIONE 2023, LNICST 515, pp. 3–26, 2025.
https://doi.org/10.1007/978-3-031-80713-8_1

Software-Defined Networking, Artificial Intelligence, and Machine Learning have a significant impact on networking. As tools to describe a large volume of problems in the networking domain, they hold significant value. These problems are currently being resolved and are expected to be resolved in the near future. In this paper, we are proposing a model to address load balancing in the intra-networking domain. We have proposed a RESTful API-based approach in this paper. A concise model for load balancing at the network gateway is defined in this work. API development is a crucial part of this proposed work. Since the inception of IoT, a large part of the Information Communication Technology community has strived to find the most efficient path to manage, configure, and monitor network performance. Unlike traditional network management, Software-Defined Networking (SDN) with Machine Learning (ML) within the networking domain makes network management and monitoring more adaptable compared to the existing state. This work primarily focuses on load balancing by incorporating the latest trends and technologies, such as SDN, RESTful API, Machine Learning, Fog Computing, Multiple Spanning Tree Protocol (MSTP), QoS Traffic Policing and Shaping, QoS order of operation, VLAN-based rate limiting, Committed Access Rate (CAR), bandwidth distribution among queues, etc. The design of the API comprises two major sections: first, the SDN-based controller handles real-time traffic information acquisition from the Gateway, and second, processing acquired information data on the server using Machine learning models to obtain upcoming thresholds, congestion, jitter, and delay. After processing the data, the model can analyze and produce results based on past experiences, leading to automatic decision-making for the re-modification of policies on a transient basis without any human intervention.

Already-introduced technologies in the field of networking, such as traditional wired LANs, Wireless LANs, Wireless sensing networks, and IoT, are deployed on a very large scale in various environments. In a broader perspective, an organization can encompass all of these technologies simultaneously, making it more complex to manage, monitor, and troubleshoot shortcomings under one domain or one single banner. It is now essential to converge all these technologies into Software-Defined Networking and Artificial Intelligence and Machine Learning-based systems, which can then compel existing technologies to reduce heterogeneity. In this paper, we emphasize the importance of handling the heterogeneity of multiple technologies using Software-Defined Networking, Artificial Intelligence, and Machine Learning to address the growing network traffic burden over gateways, maximize throughput, and balance network traffic for organizations such as ISPs, Network Administration and Monitoring Centers, or Network Operational Centers (NOCs).

2 Related Works

This research paper [1] proposes the design of a RESTful NBI for SDN applications to improve compatibility and interoperability across different SDN technologies. The research work in [2] presents an approach to ensuring fast traffic reachability in the event of a single link failure. By redirecting failed link traffic to SDN switches through pre-configured IP tunnels, the proposed approach leverages SDN capabilities to react quickly to failures. It also utilizes coordination among SDN switches to explore multiple

backup paths for efficient failure recovery, thus preventing potential congestion in the network. Additionally, it outperforms IP Fast Reroute and shortest path recalculation in terms of load balancing in the post-recovery network. The research aims to address resilience issues specific to hybrid SDN networks, contributing to improved network performance and manageability. This paper [3] proposes and explores an innovative algorithm that leverages Software Defined Networking (SDN) to dynamically manage traffic and optimize link utilization in Data Center Networks (DCNs). As DCNs become increasingly crucial in supporting computational resources and bandwidth for cloud-based applications, operational costs, link congestions, and imbalanced traffic loads pose significant challenges. The proposed algorithm addresses these issues by considering flow priorities and finding the shortest paths while calculating the cost of each link. Their work focuses on the implementation and evaluation of the algorithm in a fat-tree DCN topology. It discusses how the algorithm improves traffic management, load balancing, and resource utilization in DCNs by dynamically rerouting flows during congestion. The performance of the algorithm is measured in terms of throughput, delay, and packet loss, showcasing its effectiveness in improving overall network efficiency.

The work in [4] proposes LABERIO, a novel path-switching algorithm, to dynamically balance workloads in data center networks using the OpenFlow protocol. Unlike existing solutions that only find static routing paths during initialization, LABERIO enables dynamic traffic balancing during transmission. Through experiments on different network architectures, LABERIO demonstrates superior performance compared to traditional load balancing algorithms, reducing transmission time by up to 13%. The paper highlights the significance of load balancing in network traffic management and addresses the limitations of the OpenFlow protocol, presenting LABERIO as an innovative approach to optimizing file transmission performance, minimizing latency, and maximizing network throughput. In paper [5], the design and implementation of a web-based management system for Software-Defined Network (SDN) controllers are elaborated. This system allows network operators to control and supervise virtual network functions (VNF) provided by SDN networks. It utilizes open-source software, including Flask and Ryu frameworks, OpenvSwitch, and Mininet, and supports functions such as firewall, routing, and quality of service (QoS). The goal is to enhance network management and facilitate the dynamic configuration of network functions in a centralized and programmable manner. The work in [6] introduces RAPTOR, a REST-based API translation service for Software-Defined Networking (SDN) networks. RAPTOR allows developers to create network control software independent of specific SDN controllers by providing a unified API and translating application requests into controller-specific calls. It addresses the lack of standardization in the northbound interface (NBI) of SDN controllers, enabling interoperability and portability of code across different controllers. The feasibility and effectiveness of RAPTOR are demonstrated through its implementation and integration with various controllers in the GENI testbed.

The [7] research work describes PayLess, a monitoring framework designed for Software Defined Networking (SDN). The framework aims to simplify network monitoring tasks by providing a high-level RESTful API for flow statistics collection at different aggregation levels. It utilizes an adaptive statistics collection algorithm to balance monitoring accuracy, timeliness, and network overhead. The paper discusses the framework

of PayLess, its components, and the Monitoring API it offers for network applications. Additionally, it presents an implementation of link utilization monitoring using the proposed framework and evaluates its performance compared to other monitoring methods. The central idea revolves around the development and effectiveness of PayLess as a versatile monitoring solution for SDN environment. SEAPP is a framework designed to enhance the security of cloud computing by effectively managing application permissions and encrypting REST API calls in [8]. By implementing granular access control and authentication mechanisms, SEAPP defends against malicious attacks that exploit the openness of SDN's northbound interface. It consists of two main components: a permissions detection engine that analyzes permission manifests and bytecode to verify the authenticity and legality of application permissions, and a registration authorization engine that securely registers applications using encryption algorithms and authorizes them based on risk levels. Between the control plane and application plane, SEAPP employs a lightweight logic architecture for runtime reconfiguration and rapid deployment. The framework demonstrates strong security, effectiveness, and minimal CPU and memory overhead.

[9] presents a framework for designing the northbound API of SDN in a truly RESTful manner. It addresses design issues in RESTful networking protocols and introduces HTTP content negotiation and a hypertext-driven approach. The advantages of this approach are demonstrated through fixing design problems in existing APIs and designing a RESTful northbound API for SDN in the context of OpenStack. The framework is implemented and experimentally verified in the northbound REST API of SOX, a generalized SDN controller. The research paper [10] proposes an optimal network planning framework for Software-Defined Networks (SDNs). The framework aims to minimize the number of controllers required and reduce control traffic delay in SDNs by optimizing controller placement and control traffic forwarding paths. Research revolves around the need for control traffic balancing in Software-Defined Networks (SDNs). While existing studies have primarily focused on balancing data traffic in the data plane, control traffic balancing remains a challenging and critical issue in SDNs. Control traffic, responsible for signaling events and control plane operations, can generate a significant amount of traffic that must be effectively managed alongside data traffic. The timely delivery of control traffic is crucial for the effectiveness of routing strategies determined by the controller. Additionally, the paper highlights the importance of joint investigation of multi-controller placement and control traffic balancing for optimal network planning in SDNs.

[11] focuses on the effective utilization of Software-Defined Networking (SDN) for traffic engineering in both existing and incrementally deployed networks. The study demonstrates the benefits of leveraging SDN for traffic engineering, including improved network capacity utilization, reduced packet losses, and decreased delays. The research formulates optimization problems for the SDN controller in the context of partial SDN deployment and develops efficient algorithms to solve these problems. Through analysis and simulations, the paper showcases the performance gains achieved by these algorithms, even with limited SDN-FE deployment. The study addresses the challenges of

managing traffic effectively in hybrid networks where traditional routing protocols coexist with SDN-FEs, making valuable contributions to traffic engineering in incrementally deployed SDNs.

While traditional traffic engineering techniques have been utilized in past networks like ATM and IP/MPLS, SDN's unique characteristics require novel approaches that leverage a global network perspective to improve traffic control and management. This research paper [12] surveys SDN traffic engineering, focusing on key areas such as flow management, fault tolerance, topology update, and traffic analysis. Furthermore, it explores existing traffic engineering tools from both industry and academia and discusses open research issues essential for realizing effective SDN traffic engineering solutions.

In literature [13], an intelligent data analytics system utilizing Machine learning for the management of 5G networks is discussed. This system, known as CygNet MaSoN, aims to achieve autonomous networking capabilities by supporting advanced aggregation, analytics, and Machine learning features. It enables the detection of anomalous network behavior, degradation in network performance and service quality, as well as resource optimization. The paper describes the system's architecture, components, and three real-life use cases that showcase its effectiveness in 5G network management. The system is designed to support self-organizing and closed-loop automation functionalities expected in autonomous 5G networks. Research [14] aims to provide a comprehensive overview of recent efforts to incorporate AI into SDN, exploring its application areas and potential benefits. By separating the control plane from the data plane and utilizing a central controller as the brain of the network, AI-based techniques can enhance decision-making, traffic engineering, network optimization, security, and other aspects of SDN architecture.

In contrast to the above paragraphs, our work emphasizes QoS with load balancing with the integration of different recently developed technologies. In the above-mentioned works, we studied the integration of recently developed AI and Machine learning applications in the field of networking. Some of the above-mentioned works focus on embedding AI into the SDN controller, while others concentrate on integrating Machine learning into 5G network management systems to achieve autonomous networking capabilities. One of the above-mentioned works describes Virtual Network Functionality (VNF) to manage SDN virtually from a remote location. Our work involves processing information on a server using Machine learning models to predict upcoming future network datarates. We employ multiple machine learning techniques to learn network traffic patterns and predict future datarates. We use LSTM for time series prediction in combination with Random Forest Regression Ensemble to optimize LSTM predictions for more accurate results. This methodology distinguishes our work from previously conducted research. In our work, we bring together various technologies to achieve our research goal. We intend to extend our work in this area by integrating technologies such as RESTful API, SDN, and AI. In previous related works, there have been fewer research efforts aimed at integrating different technologies for the development of autonomous networks, virtualized network functionalities, or the merger of AI and Machine learning capabilities into traditional networks or SDN. For this purpose, we aim to develop the Ryu SDN controller, as mentioned in this work. The integration of the Ryu SDN controller with RESTful API enables the transfer of information data to the server for processing. On

the server, we have enabled a Machine learning model to process the data and make predictions, as demonstrated in our work in the later sections. After the completion of processing data on the Machine learning model, the autonomous policy modification and implementation phase will be initiated, making our work unique compared to the above-mentioned works. In this work, we have developed a Machine learning model to predict future datarates on specific interfaces. In light of the above-mentioned works, where several approaches have been developed and functional in the current state using RESTful API, SDN, and AI Machine learning capabilities, our endeavor is to develop an integrated autonomous network management system.

3 Problem Formulation, Topology and Model

We have developed a topology for a Border Gateway traffic system. In this topology, we propose a network consisting of Routers/Switches at the network's edge. The border gateway device is connected to a server. Our primary concern is balancing the load of network traffic towards the gateway. In this research work, we collected traffic data information from physical interfaces only. We intend to extend this work to include VLANs and specific IP addresses assigned to clients present in the network. For this work, we have only considered traffic flow information from physical interfaces for experimentation. On the other hand, we have described our future interests in this work. We intend to expand upon this work and the topology we have already described. In Fig. 1, we depict the topology designed to address load balancing and QoS (Quality of Service). In Fig. 2, we define a framework that includes interfaces, VLANs, and individual IP addresses for capturing traffic flow. Both the topology and framework we have designed contain the same elements and components, whether covered in this work or as extensions to this work. The topology consists of several steps to achieve load balancing within its intra-network domain, efficiently allocating bandwidth to the egress interface and balancing the traffic load of the network. The proposed topology includes several Virtual Local Area Networks (e.g., VLAN10-VLAN50). These VLANs contain pools of IP addresses with defined/default data rates to access the internet. In other words, each IP address, VLAN, and physical interface (port) on a Gateway has limitations on traversing traffic, either as defined by the administrator to limit the traffic rate. In this work, we cover information collection from the physical interfaces to develop a machine learning-based prediction model using previous data rate records. In the extension of this work, we will implement the same methodology to collect VLAN and separate client IP address traffic data rate information.

The Fig. 1 is topology for this research work to resolve the problem of QoS (Quality of Service) load balancing by implementing Machine learning model. We design this topology to implement Machine leaning model on a network. Our aim is to build a network management pliable as compare to current state. By reading traffic flow statistics we can analyze traffic flow patterns. We intend to use machine learning to prediction future incoming traffic load on network. By applying Load balance technique enabled with machine learning model capabilities, we can resolve upcoming future network threshold, congestion, jitter, latency etc. for better QoS (Quality of Service). Machine learning model learns from past traffic information data.

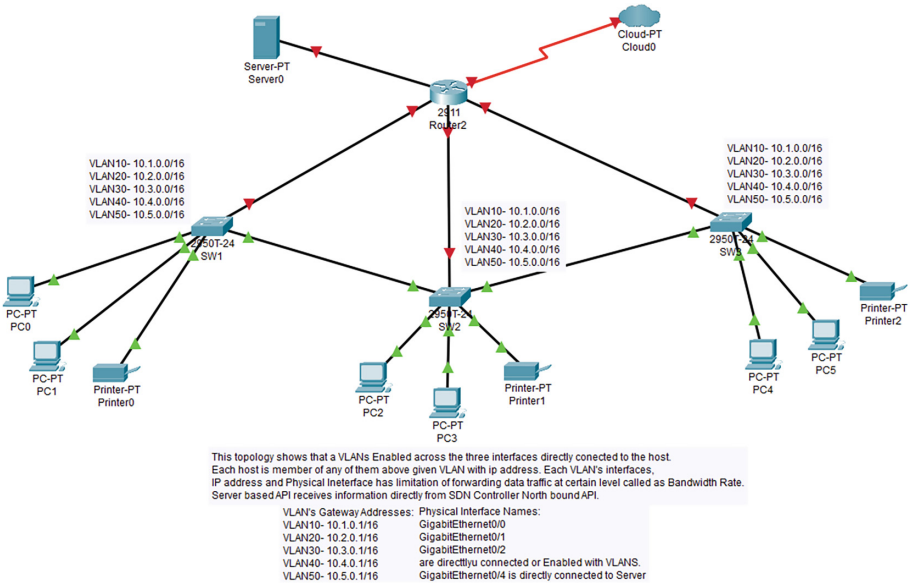

Fig. 1. Topology of Software Defined Network

Thus, it can predict future values by which network policies can be automatically modified. We adopt several previously developed renowned protocol strategies as references for this research work: [15, 16] QoS, Traffic Policing and Shaping (associating the configured QoS settings with the appropriate VLANs so that the rate limiting rules are enforced on the traffic passing through those VLANs), [17] VLAN-based rate limiting, [18] Committed Access Rate (CAR) - the primary purpose of which is to prevent traffic congestion on critical links by limiting the amount of traffic that can be sent through those interfaces, bandwidth distribution among queues, and Multiple Spanning Tree Protocol (MSTP). As we have discussed, our topology's relevance is relatively closer to MSTP-related topology. MSTP, also known as IEEE 802.1s, extends the concepts of STP and RSTP to support multiple VLANs (Virtual LANs) within a network. MSTP allows defining multiple spanning trees in the network, where each spanning tree corresponds to a specific group of VLANs. This allows for better utilization of network resources by providing separate spanning trees for different VLANs, reducing the number of blocked ports, and increasing available bandwidth.

Research [19] proposes a scheme called the BMST (Best Multiple Spanning Tree) algorithm to improve the load balancing and traffic engineering capabilities of Ethernet networks using the IEEE Multiple Spanning Tree Protocol (MSTP). The algorithm considers all possible edge-disjoint spanning trees and VLAN groupings, aiming to find the best solution based on load balancing on links and switches, as well as the shortest path selection [20]. The proposed approach aims to achieve faster convergence time, reusability of VLAN tags, protection from failures, and optimal broadcast domain size. By dividing the network into smaller MST regions based on VLANs, the algorithm facilitates better resource utilization, localization of failures, and faster recovery

[21]. Focuses on utilizing the IEEE 802.1S Multiple Spanning Tree Protocol (MSTP) to enhance load balancing and resilience in Ethernet carrier networks. Specifically addressing the single-region MSTP case, the paper explores the benefits of MSTP in allowing network operators to define different Spanning Tree instances and assign VLANs without constraints, thereby improving network load balancing.

[22] We enable Artificial intelligence and Machine learning capabilities on Software-Defined Network. For the collection of traffic records, the RESTful API plays a crucial part in bringing traffic information from the SDN controller to the server placed at a remote location for processing information data. The server is responsible for gathering real-time information about all IP addresses currently active in a group of VLANs, collective traffic traverse information within VLANs, as well as collecting information about each GigabitEthernet physical interface data traverse information. For this purpose, we have developed an API endpoint on both sides, the SDN controller, and the server, which is responsible for collecting information from the remote Border Gateway device (such as router/switch). Each endpoint is responsible for collecting information received from the SDN controller-based Northbound API. The collected information is then stored on a server for processing. The framework diagram in Fig. 2 elaborates the RESTful-SDN framework.

Recent trends are utterly divergent, shifting from distributed computing to centralized control systems in the area of network control, management, and monitoring systems. In contrast, SDN decouples the control plane from the data plane. In other words, in our research work, we aim to leverage SDN technology to implement a Machine learning-based centralized network traffic policies management system. With the emergence of Artificial Intelligence and Machine Learning, numerous novel strategies and techniques have been introduced for network management and monitoring. We have adopted MSTP techniques, QoS rate limiting on IP, VLAN, and Interface such as CAR, QoS traffic policing and shaping, unused bandwidth utilization, etc., to implement in SDN by using a RESTful API with a server enabled with Machine learning-based future policy mapping with respect to time.

Henceforth, Henceforth, our topology elaborates (with the Ryu controller acting as a Load balancer and incorporating QoS) implementation of Machine Learning to the Edge of the Network where the Border Gateway device is connected to the server. In Fig. 3, the structure of the traditional architecture of the Network is compared with the SDN architecture defined in the SDN-RESTful based framework in Fig. 2.

In this research work, we conduct network traffic traversal prediction by utilizing previously recorded traffic records on the SDN-connected server. In Fig. 2, the traversal data recording process is implemented in three segments: (1) physical interfaces, (2) VLAN interfaces, and (3) individual IP addresses with timestamps. Based on traffic information, our aim is to predict future values for each physical interface, VLAN interface, and IP address, respectively. These predictions for the future have the potential to redefine network policies, reducing network overhead, congestion, and jitter effects, which are directly proportional to achieving steady, stable, and smooth traffic traversal. Such steady and smooth network traffic traversal is undoubtedly a pressing need. Performing statistical data analysis through deep learning models and machine learning models is a sophisticated approach that has recently gained significant traction among

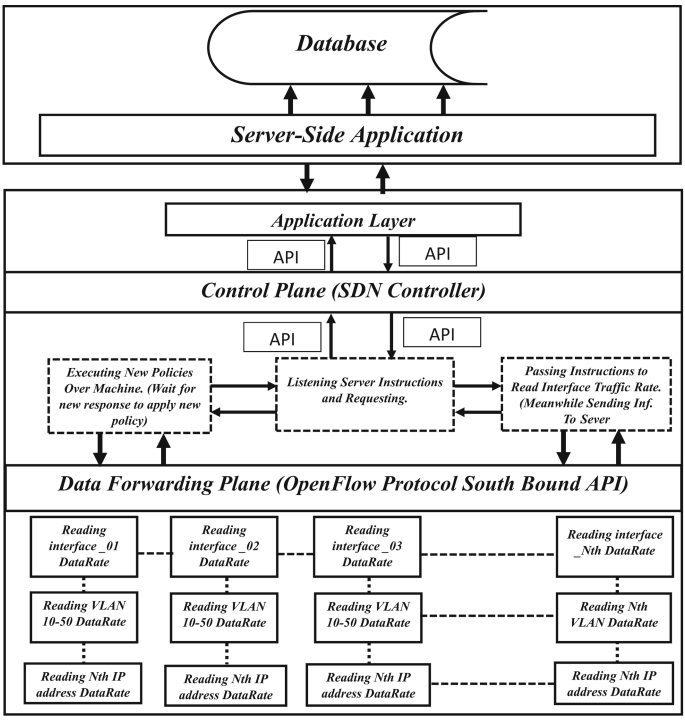

Fig. 2. Framework diagram of SDN with RESTful API

network specialists, scientists, and researchers. The issue of network traffic flow can be more effectively resolved by implementing this methodology of Machine Learning on network traffic flow. For example, as previously discussed, Vlan-based rate limiting, Committed Access Rate (CAR), Traffic Policing and Shaping, MSTP, RSTP, etc., in the earlier paragraphs. There is no research work that addresses Machine Learning to predict these issues in advance and resolve them with minimal human intervention. This demonstrates the novelty of our work compared to previous efforts.

First, we can efficiently utilize the dedicated data rate allocated to our network. Machine Learning can reduce human intervention in scheduling static and dynamic bandwidth assignments, leading to reduced errors. This methodology improves the bandwidth sharing mechanism efficiently. Second, Machine Learning model predictions can anticipate upcoming network congestion. Third, due to early alerts from Machine Learning implemented on network traffic, bottleneck issues can be prevented by using dynamic bandwidth sharing.

4 Experimental Work

In this section we describe our work. First, we have collected traffic information in the Laboratory of Information Science and Engineering, Tianjin university of Technology and Education. For this experiment section we cover an experiment on the basis

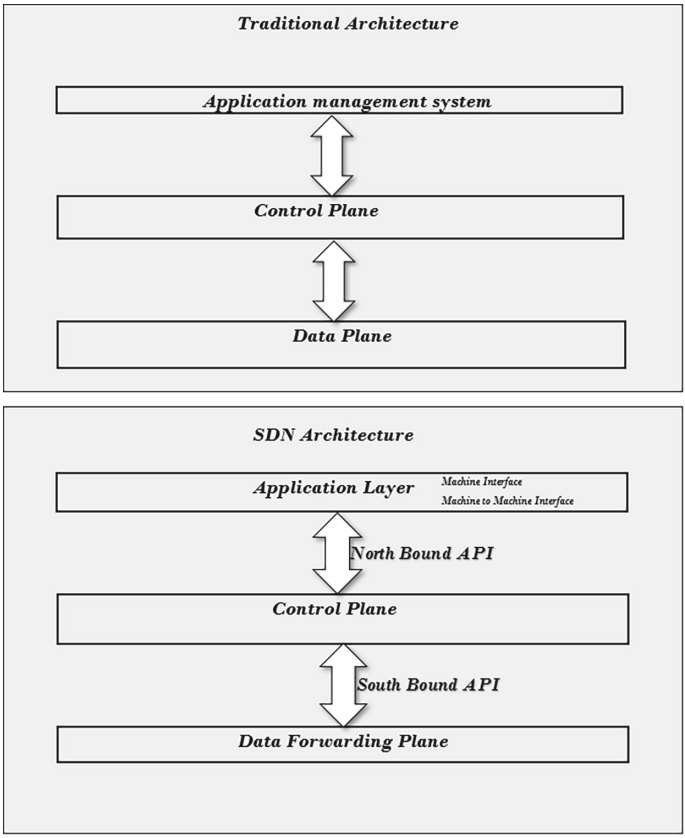

Fig. 3. Traditional and SDN Architecture

of collected information data. The data collection method we adopted in this paper is simulation-based data acquisition. We used VMware Workstation for data collection. In the simulation environment we created virtual router on a virtual host Machine by using Windows Router and Remote Access Service (RRAS), Server Manager. The given virtual interface in the virtual environment is GigabitEthernet. Each interface of each virtual Machine has gateway on a windows server Machine, where we have deployed windows routing and remote service for routing among different virtual Machines. We use subnets for each host Machine to communicate with each other in the virtual environment. Each subnet in the virtual environment routes traffic through virtual interfaces. Although, traffic generation and data acquisition in virtual environment has limitations. For traffic generation we used iPerf to generate traffic across required network. For this purpose, we installed iPerf on each Machine which is subject to communicate with other Machines with different subnet. Henceforth, we used python script to capture traffic information of host Machine by reading those interfaces called as Interface01, Interface02 and Interface03 respectively. In Figs. 4, 5, 6, 7, 8, we recorded traffic information from multiple interfaces GigabitEthernet called as Interface01, Interface0 and

Interaface03 respectively. we created a table for each interface called with accordance to the interface's names interface01, interface02 and interface03 to save the record of traffic passing through over the network gigabit interface.

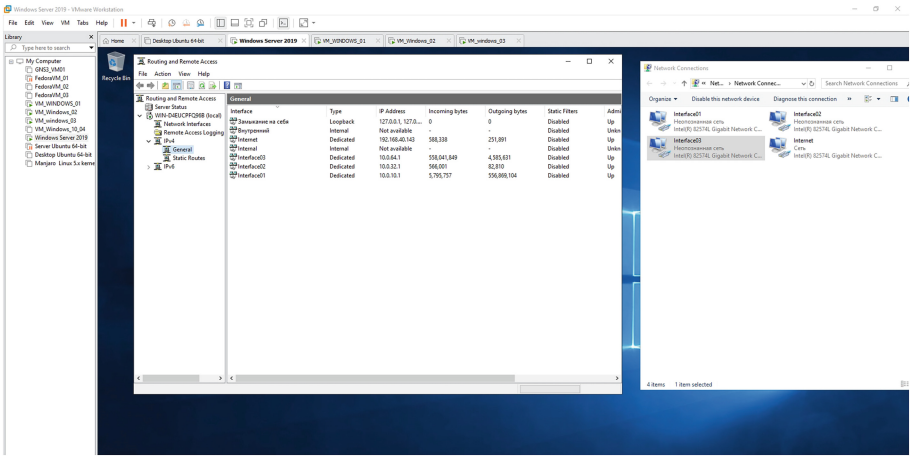

Fig. 4. Configuration of Multiple Interfaces for Traffic traverse

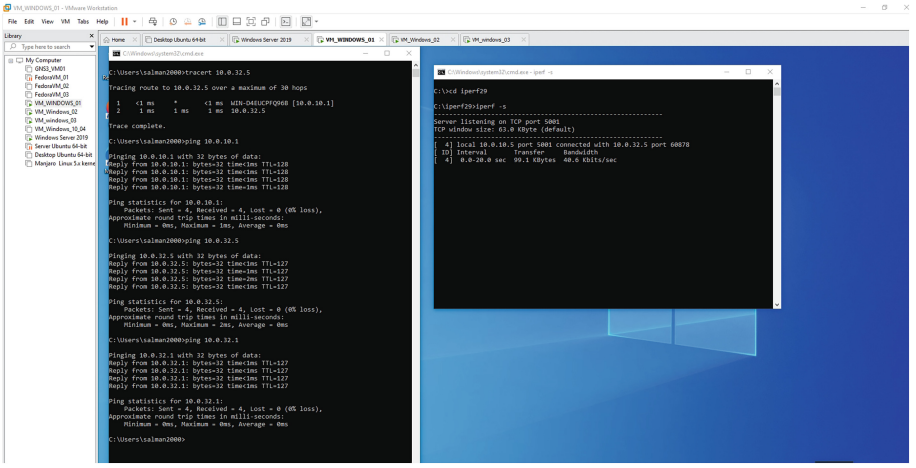

Fig. 5. First client virtual machine communication through Interface-01.

We recorded inbound and outbound traffic as well as aggregate data rates over a particular interface. In Figs. 9 and 10, we present a flow diagram of our framework in a step-by-step manner. Each step encompasses an operation that must be completed before proceeding to the next step within the entire process. We designed and developed two Machine learning models for conducting our experiments, both of which are aimed at predicting future values. These Machine learning models are LSTM (Long Short-Term

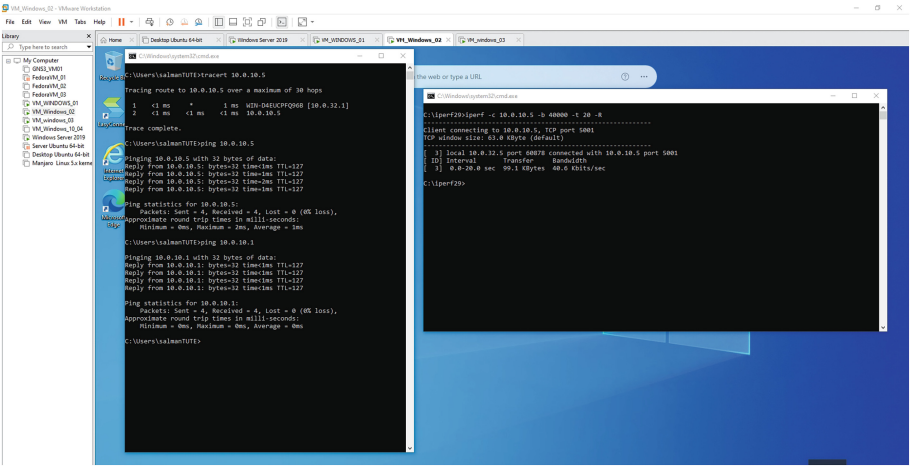

Fig. 6. Second client virtual machine communication through Interface-02.

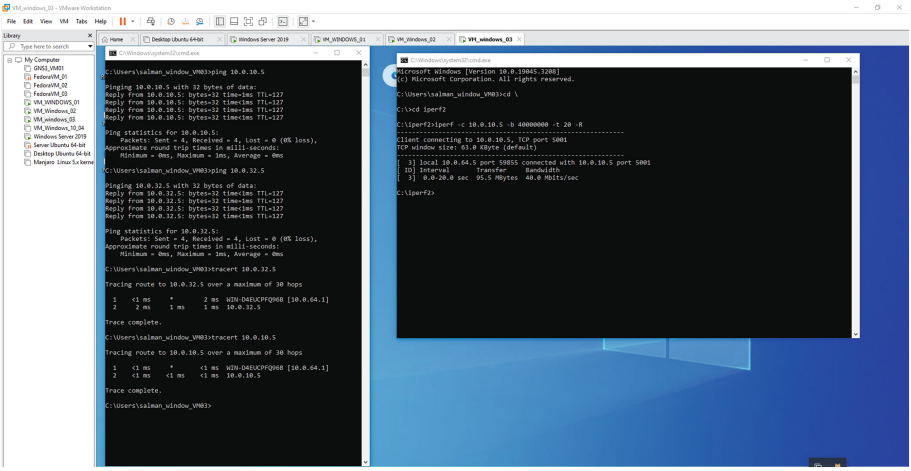

Fig. 7. Third client virtual machine communication through interface-03

Memory) and Random Forest Regression Ensemble. We implemented LSTM on each interface's traffic records, experimenting with various tuning parameters in the model to predict future values of interface traffic. Our research work proceeded in a stepwise fashion, and subsequently, we obtained results (prediction values) from LSTM. The prediction data obtained was then further processed in the Random Forest Regression Ensemble model to enhance precision and robustness for decision-making purposes. To facilitate this process, we designed and developed a workflow for our research work. Our experiments in this research work followed the defined workflow outlined below in this section.

Pseudocode 1: Reading N interface, store Data to a database as D dataset Matrix

```
Input: input parameter interface_name[n], IP[n] address
Output: 'D' dataset → Database

procedure interfaceStat_Search[list]
  loop: for each item in list
    Declaration
      Initialize empty list Interface_name
      Initialize empty list IP_address

  Input num_interface
  Initialize iter to 0
  loop: while iter <= num_interface
    Input interface_name[n]
    Input IP[n] address

    If interface_name[n] is in Interface_name and IP[n] address is in IP_address
      loop: while True
        procedure interface(interface_name[n], IP[n])
          Begin
            Call function get_time()
            Call function get_sentbits ()
            Call function get_recvbits ()
            Call function total_sum(get_sentbits (), get_recvbits ())
            Read data from the database
            Get columns: datetime, received, sent, total
            Append columns to dataset D matrix
            Save record to the database
          end procedure
        end loop
        break;
      end loop
    end loop
  end procedure
```

Fig. 8. Pseudocode to read traffic information from interfaces

4.1 LSTM Based Prediction on Multiple Interface

LSTM (Long Short-Term Memory) networks is a kind of recurrent neural network (RNN) that are well-suited for sequence modeling tasks, including time series analysis and prediction. LSTM (Long Short-Term Memory) networks are commonly used in the field of sequence modeling, including time series forecasting tasks such as network traffic prediction. LSTM networks are a type of recurrent neural network (RNN) designed to capture long-term dependencies and patterns in sequential data.

Splitting the Data. Preprocess the data by removing any irrelevant information or noise. This may include removing headers, normalizing timestamps, and eliminating duplicate

Pseudocode 2: Flow of Program

```
Import the necessary modules and functions from the mainpack package.

Import the multiprocessing module.
Import the time module.

Define a function named progflow():

  loop: while True

    Create three multiprocessing processes:
      - p1 for calling the function start01.inf_01()
      - p2 for calling the function start02.inf_01()
      - p3 for calling the function start03.inf_01()

    Start the three processes (p1, p2, p3).

    Wait for all three processes to finish (p1, p2, p3) using join().

    Get the current time and store it in a variable endtime.

    loop: for y in range (1)
      Create three multiprocessing processes:
        - p5 for calling the function lstm()
        - p6 for calling the function lstm02()
        - p7 for calling the function lstm03()

      Start the three processes (p5, p6, p7).

      Wait for all three processes to finish (p5, p6, p7) using join().

      Get the current time and store it in a variable endtime.

      loop: for z in range (1)
        Create three multiprocessing processes:
          - p8 for calling the function rfr01()
          - p9 for calling the function rfr02()
          - p10 for calling the function rfr03()

        Start the three processes (p8, p9, p10).

        Wait for all three processes to finish (p8, p9, p10) using join().

If __name__ == '__main__':
  Call the function progflow ().
```

Fig. 9. Pseudocode of Program Flow

or incomplete entries. Split the preprocessed data into two phases: Training and Testing. The training phase data will be used to build the sequence formation model, while the

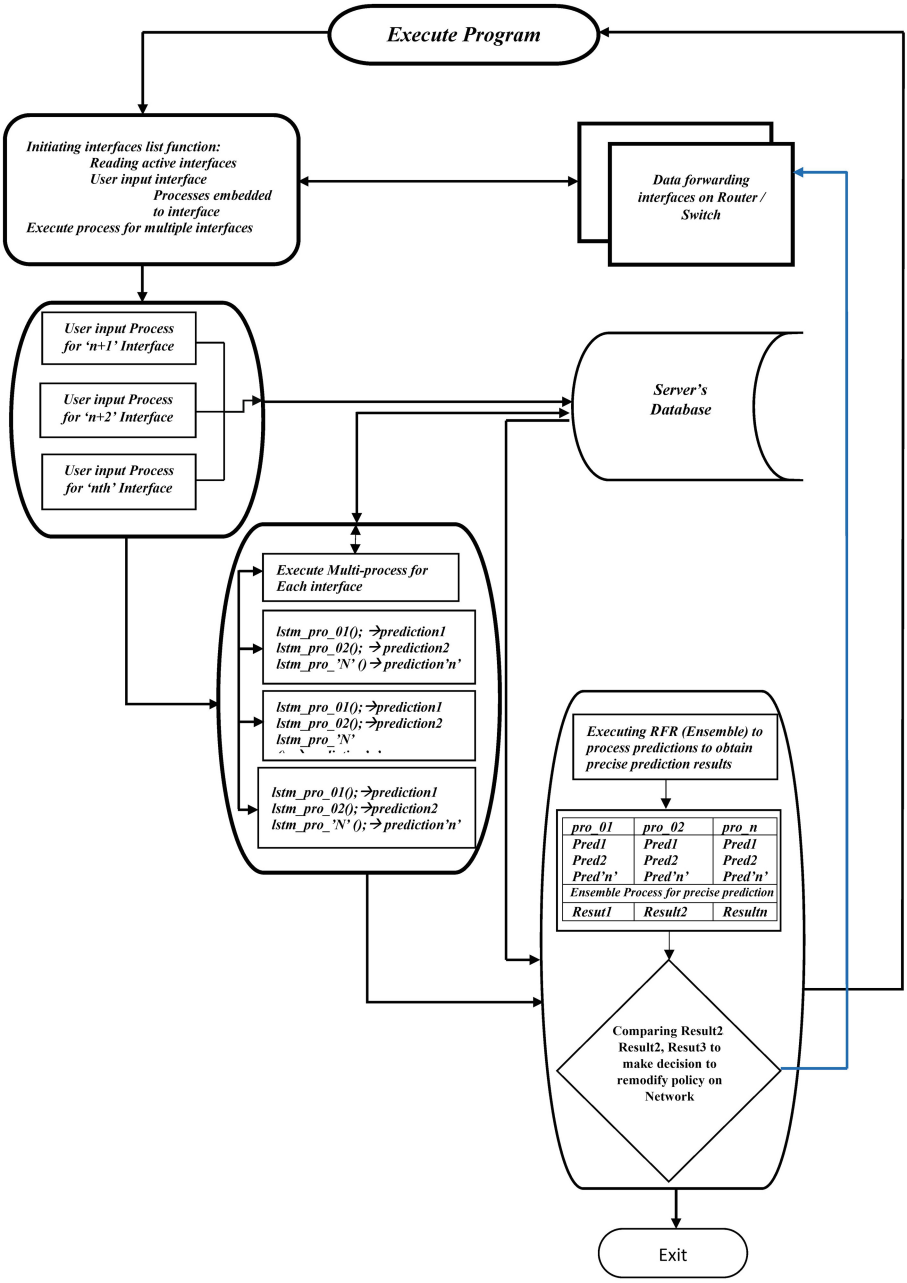

Fig. 10. Flow diagram of framework

testing phase data will be used to evaluate the model's performance.

[477024 rows x 3 columns]
trainX shape = (381616, 3, 3)
trainY shape = (381616, 1)
estX shape = (95405, 3, 3)

We have selected supervised learning techniques in which the dataset is labelled dataset. Feature can be extracted or observed. For feature engineering we used Sklearn (Sci-Kit Learn) for data preprocessing, normalization and feature engineering by using StandardScaler module. We selected interface Tx (receiving rate), Tr (sending rate) and total DataRate with respect to time. In our work we consider the input and output feature where receiving traffic rate, sending traffic rate and sum of both receiving DataRate ad sending DataRate named as total DataRate. Whereas, in linear regression supervised learning model works on the principle of, $y = wx + b$ for simple linear regression if there is multiple input variable then $y = w1x1 + w2x2 \ldots. wn.xn + b$, that means the feature correlations and dependent variable. Meanwhile m, m1, m2 are the correlation coefficient of independent variable of *x, x1 or x2 …. xn,* which describe the impact of coefficient to best fit the model. The output variable *y* is dependent on multiple input variable of *x*. In this case *y^* is the output prediction variable. The filter techniques are dropping column method in python to select relevant feature. We can apply necessary feature engineering techniques such as filtering to extract relevant information data columns from the network traffic dataset if there is necessary. This may involve extracting protocol types, packet sizes, source/destination IP addresses, or any other relevant attributes that can help capture the behavior and patterns in the sequences. Herein, we extract some feature from our dataset to train model to predict (*y^*) upcoming traffic for a particular interface. Once formed the sequences and extracted the relevant features, train a Machine learning or deep learning model. Common models used for sequence analysis include Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM) networks, or Transformer models.

Evaluate the trained model's performance using the testing phase data. This step involves feeding the testing sequences into the model and analyzing its predictions against the ground truth or expected outcomes. We use various evaluation metrics such as Mean Square Error (MSE), Root Mean Square Error (RMSE). Mean Absolute Error (MAE) to assess the model's effectiveness in capturing the patterns in the network traffic information data. Table 1 above in this section to elaborate effectiveness of our model performance by using above given performance metrics.

Sequential Modeling and Layering. By splitting the data into two phases and following the sequence formation process, effectively convert network traffic data into sequences and analyze them using Machine learning or deep learning techniques. Layers can be seen below in Fig. 11.

Sigmoid (Activation Function). It is mainly used for probability-based prediction modeling. Sigmoid is often preferred choice for sequential modeling, `sigmoid(x)= 1 / (1+ exp^(-x))`. Sigmoid activation function works in between 0, 1. Because the probability lies between 0 to 1.

Table 1. Model Evaluation by using MAE, MSE and RMSE.

	Interface 01						*Interface 02*						*Interface 03*					
	Epoch up to 50 of interface-01 for 1st prediction						*Epoch up to 100 interface-02 for 1st prediction*						*Epoch up to 200 of interface-03information for 1st prediction*					
	loss mse	*val_loss mse*	*loss mae*	*val_loss mae*	*loss rmse*	*val_loss rmse*	*loss mse*	*val_loss mse*	*loss mae*	*val_loss mae*	*loss rmse*	*val_loss rmse*	*loss mse*	*val_loss mse*	*loss mae*	*val_loss mae*	*loss rmse*	*val_loss rmse*
Epoch 01	0.9433	0.9269	0.7154	0.7074	0.9712	0.9628	0.9421	0.9280	0.7154	0.6953	0.9706	0.9633	0.9439	0.9273	0.7163	0.7015	0.9715	0.9630
Epoch 02	0.9285	0.9177	0.7066	0.6906	0.9636	0.9580	0.9279	0.9214	0.7060	0.6892	0.9633	0.9599	0.9368	0.9259	0.7077	0.6758	0.9679	0.9623
Epoch 03	0.9221	0.9306	0.7033	0.6915	0.9603	0.9647	0.9221	0.9188	0.7035	0.7127	0.9603	0.9586	0.9264	0.9166	0.7044	0.6885	0.9625	0.9574
Epoch 04	0.9191	0.9145	0.7019	0.6793	0.9579	0.9563	0.9201	0.9217	0.7026	0.6820	0.9592	0.9601	0.9211	0.9181	0.7029	0.6901	0.9597	0.9582
Epoch 05	0.9155	0.9182	0.7003	0.6853	0.9568	0.9582	0.9201	0.9150	0.7013	0.7008	0.9578	0.9566	0.9347	0.9254	0.7026	0.6780	0.9668	0.9620
Epoch 46	0.9089	0.9089	0.6966	0.6895	0.9534	0.9533	0.9043	0.9104	0.6952	0.6805	0.9509	0.9541	0.9075	0.9052	0.6964	0.6787	0.9526	0.9514
Epoch 47	0.9082	0.9053	0.6964	0.6871	0.9530	0.9515	0.9062	0.9037	0.6960	0.6766	0.9519	0.9506	0.9065	0.9125	0.6965	0.6831	0.9521	0.9552
Epoch 48	0.9087	0.9060	0.6968	0.6835	0.9532	0.9519	0.9051	0.9049	0.6956	0.6806	0.9514	0.9513	0.9069	0.9030	0.6960	0.6778	0.9523	0.9502
Epoch 49	0.9079	0.9086	0.6963	0.6782	0.9528	0.9532	0.9048	0.9021	0.6954	0.6919	0.9512	0.9498	0.9071	0.9039	0.6962	0.6821	0.9524	0.9508
Epoch 50	0.9058	0.9057	0.6958	0.6938	0.9518	0.9517	0.9032	0.9073	0.6948	0.6972	0.9504	0.9525	0.9089	0.9025	0.6961	0.6817	0.9533	0.9500
	Epoch up to 100 of interface-01 for 2nd prediction						*Epoch up to 100 of interface-02 for 2nd prediction*						*Epoch up to 100 of interface-03 for 2nd prediction*					
Epoch 01	0.9480	0.9214	0.7140	0.6877	0.9736	0.9599	0.9517	0.9298	0.7158	0.6999	0.9756	0.9643	0.9468	0.9402	0.7161	0.7242	0.9730	0.9696
Epoch 02	0.9292	0.9268	0.7056	0.6677	0.9639	0.9627	0.9307	0.9261	0.7072	0.6872	0.9647	0.9624	0.9291	0.9254	0.7083	0.6932	0.9639	0.9620
Epoch 03	0.9263	0.9188	0.7033	0.6866	0.9624	0.9585	0.9309	0.9195	0.7053	0.7013	0.9648	0.9589	0.9228	0.9222	0.7039	0.6741	0.9606	0.9603
Epoch 04	0.9292	0.9245	0.7025	0.6914	0.9639	0.9615	0.9237	0.9217	0.7032	0.6861	0.9611	0.9600	0.9185	0.9164	0.7020	0.6986	0.9584	0.9573
Epoch 05	0.9191	0.9134	0.7013	0.6951	0.9587	0.9557	0.9207	0.9146	0.7018	0.6896	0.9595	0.9564	0.9173	0.9136	0.7011	0.6969	0.9578	0.9558
Epoch 96	0.9062	0.9072	0.6958	0.6856	0.9520	0.9525	0.9072	0.9013	0.6945	0.7008	0.9525	0.9493	0.9039	0.8991	0.6943	0.6817	0.9507	0.9482
Epoch 97	0.9073	0.9018	0.6957	0.6752	0.9525	0.9496	0.9052	0.9022	0.6942	0.6784	0.9514	0.9498	0.9050	0.8994	0.6941	0.6740	0.9513	0.9484
Epoch 98	0.9078	0.9076	0.6957	0.6906	0.9528	0.9527	0.9090	0.9003	0.6954	0.6834	0.9534	0.9489	0.9022	0.8995	0.6938	0.6821	0.9498	0.9484
Epoch 99	0.9098	0.9036	0.6961	0.6762	0.9538	0.9506	0.9095	0.9010	0.6951	0.6870	0.9537	0.9492	0.9080	0.9113	0.6947	0.6833	0.9529	0.9546
Epoch 100	0.9048	0.8994	0.6952	0.6870	0.9512	0.9483	0.9079	0.9051	0.6948	0.6944	0.9529	0.9514	0.9030	0.9014	0.6942	0.6865	0.9503	0.9494
	Epoch up to 200 of interface-01 for 3rd prediction						*Epoch up to 200 of Interface-01 for 3rd prediction*						*Epoch up to 200 of interface-01 for 3rd prediction*					
Epoch 01	0.9474	0.9349	0.7170	0.6993	0.9734	0.9669	0.9486	0.9221	0.7147	0.7031	0.9740	0.9603	0.9456	0.9323	0.7149	0.6808	0.9724	0.9656
Epoch 02	0.9335	0.9662	0.7102	0.6997	0.9662	0.9640	0.9267	0.9232	0.7054	0.6819	0.9627	0.9608	0.9351	0.9277	0.7066	0.6719	0.9670	0.9632
Epoch 03	0.9225	0.9605	0.7043	0.6770	0.9605	0.9586	0.9225	0.9200	0.7033	0.6731	0.9605	0.9592	0.9260	0.9155	0.7040	0.6946	0.9623	0.9568
Epoch 04	0.9206	0.9595	0.7024	0.7083	0.9595	0.9662	0.9184	0.9217	0.7019	0.6776	0.9583	0.9600	0.9215	0.9152	0.7024	0.6908	0.9600	0.9567
Epoch 05	0.9163	0.9572	0.7010	0.6882	0.9572	0.9566	0.9178	0.9167	0.7012	0.6978	0.9580	0.9574	0.9205	0.9165	0.7018	0.6898	0.9594	0.9573
Epoch 196	0.9059	0.9048	0.6943	0.6989	0.9518	0.9512	0.9060	0.8983	0.6940	0.6788	0.9518	0.9478	0.9121	0.9026	0.6961	0.6830	0.9550	0.9501
Epoch 197	0.9031	0.9055	0.6929	0.6669	0.9503	0.9516	0.9045	0.9155	0.6934	0.6907	0.9511	0.9568	0.9116	0.9083	0.6959	0.6811	0.9548	0.9531
Epoch 198	9055	0.9047	0.6937	0.6799	0.9516	0.9511	0.9034	0.8985	0.6937	0.6829	0.9505	0.9479	0.9110	0.8997	0.6950	0.6894	0.9545	0.9485
Epoch 199	0.9110	0.9132	0.6947	0.6933	0.9544	0.9556	0.9048	0.9075	0.6930	0.6868	0.9512	0.9526	0.9087	0.9043	0.6943	0.6781	0.9533	0.9510
Epoch 200	0.9062	0.8980	0.6937	0.6798	0.9519	0.9476	0.9024	0.8985	0.6928	0.6870	0.9500	0.9489	0.9090	0.9014	0.6946	0.6769	0.9534	0.9494

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 3, 64)             17408
lstm_1 (LSTM)                (None, 32)                12416
dropout (Dropout)            (None, 32)                0
dense (Dense)                (None, 2)                 66
dense_1 (Dense)              (None, 1)                 3
=================================================================
Total params: 29,893
Trainable params: 29,893
Non-trainable params: 0
_________________________________________________________________
```

Fig. 11. Sequential Modeling and Layering

Dense Layer. It allows the model to learn complex patterns by combining information from all input features. The dense layer is suitable for regression and time series forecasting. In the context of neural networks, a dense layer is a fully connected layer where each neuron is connected to every neuron in the previous layer. We use Dense layers are used to learn complex patterns in the data by performing matrix multiplications and applying an activation function to the weighted sum of inputs.

Dropout Layers. We used a Dropout layer which is a regularization technique used to prevent overfitting in deep neural networks. It randomly sets a fraction of input units to zero during the training process, which helps the model to become more robust by reducing the reliance on specific neurons which results that model learn more robust features.

Adam Optimizer. Adam (Adaptive Moment Estimation) is an optimization algorithm commonly used in deep learning models to update the weights of neural networks during training. It combines the benefits of two other optimization techniques, AdaGrad and RMSProp, and adapts the learning rate for each parameter individually to accelerate convergence and improve training efficiency. We added the Adam optimizer in LSTM time series forecasting to employ several advanced techniques to improve the model's performance and results. Learning Rate Scheduling Adjust the learning rate during training. Early Stopping: Monitoring a validation metric during training, such as validation loss or accuracy, and stopping training when the metric stops improving can prevent overfitting. Monitoring a validation metric up to optimal number of epochs to train the model. Gradient Clipping: LSTM models can suffer from exploding gradients, especially in deep networks. Adam optimizer helps limit the magnitude of gradients, preventing numerical instability during training by applying gradient clipping.

LSTM Results on Epoch 50 at interface-01 (Figs. 12, 13 and 14). By processing interface collected information data into LSTM we have obtained results in form of predictions. Predictions obtained for LSTM may subject to use in further processing to improve and optimize prediction to sustain quality of prediction and model accuracy. Note that from Fig. 12, 13 and 14, we have shown the LSTM training, test, evaluation metrics mse, rmse and predictions on given dataset to the LSTM. Figure 12 shows that LSTM learning from the past data and tries to reach a minimal optimum point. Meanwhile, In Fig. 13 shows that model evaluation metrics graph. In Fig. 14 we have shown obtained predictions from LSTM.

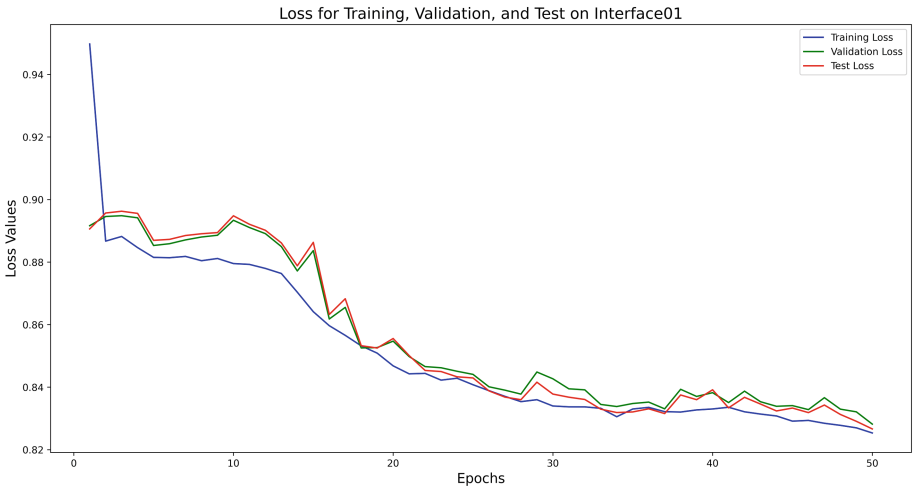

Fig. 12. Model training and convergence

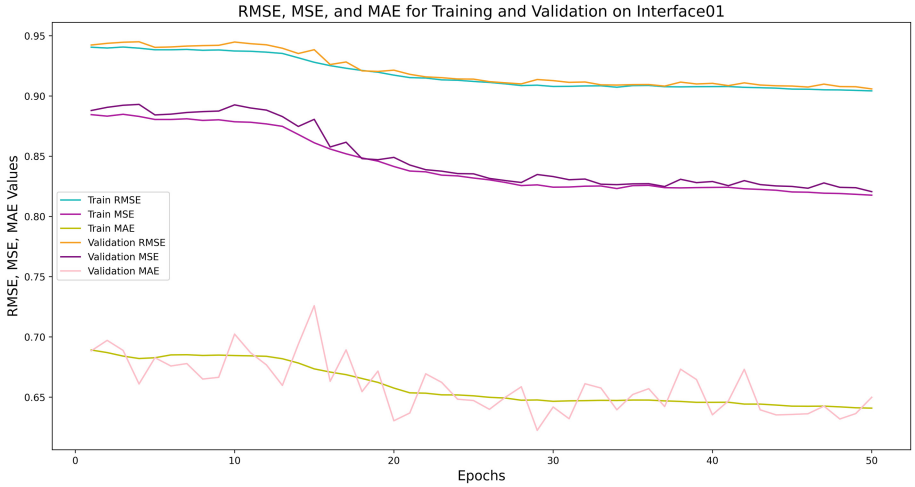

Fig. 13. MSE, MAE and RMSE for model evaluation

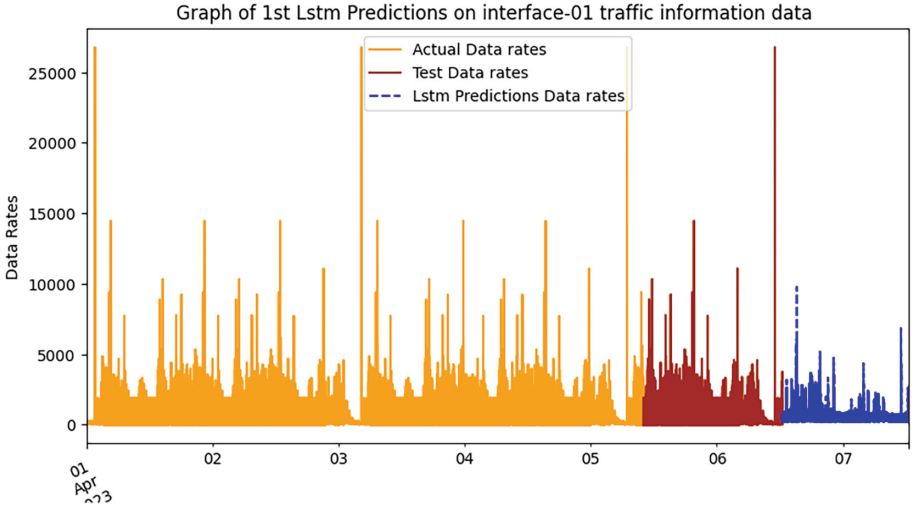

Fig. 14. LSTM Predictions on Interface-01

4.2 Optimization by Random Forest Regression Ensemble

Random Forest Regression Ensemble is a valuable tool in the field of Machine learning for regression analysis. By combining multiple decision trees, it can effectively handle complex relationships and provide accurate predictions. The main idea behind Random Forest Regression Ensemble is to build a multitude of decision trees and then combine their predictions to obtain a more robust and accurate result. In this ensemble method, each decision tree is constructed using a random subset of the raining data and a random subset of the input features. We use Random Forest Regression Ensemble in our work

to obtain robust predictions. There are some steps given below we have conducted in this work to obtain more robust results.

Reading Predictions from Database. It reads predictions from three separate table into separate pandas Data Frames.

Merging Predictions. The predictions from the three Data Frames are merged on the "date time" column using an "outer" join method to create a single data Frame. We join all LSTM prediction for future data rate with timestamps and timestamp considered as index column.

Generating Mean of All Predictions. We calculated the mean of three prediction columns row-wise to creates a new 'Average' column in the data Frame to train the model. The ensemble model method is to take mean of all given predations from different models or multiple predictions from a model.

Generating Periodic Mean of All Predictions. It calculates the mean of the 'Average' column up to each row, storing the results in a new 'Periodic Mean' column. This column we added to our model to understand the periodic declination and rise of prediction. Thus, this will provide model the pattern of prediction.

Splitting Data into Training and Testing Sets. The data is split into training and testing sets using the train_test_split function from scikit-learn.

Creating and Training the Random Forest Regression Model. A Random Forest Regressor model is created with specified hyperparameters and trained using the training data. We use n_estimators, max_features, square error as criterion, n_jobs, oob_score, random_state, verbose, min_samples_leaf, bootstrap. For training model, we consider three predictions columns of as input feature and output feature we selected as Mean of all predictions, meanwhile date and time selected as index.

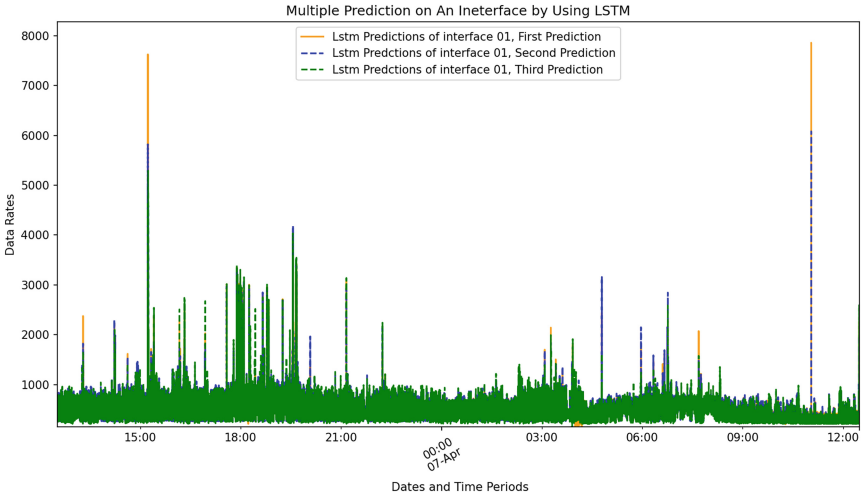

Fig. 15. Line graph of multiple prediction collected from interface-01 by using LSTM.

Making Predictions on New Data. The code generates new dates and predictions

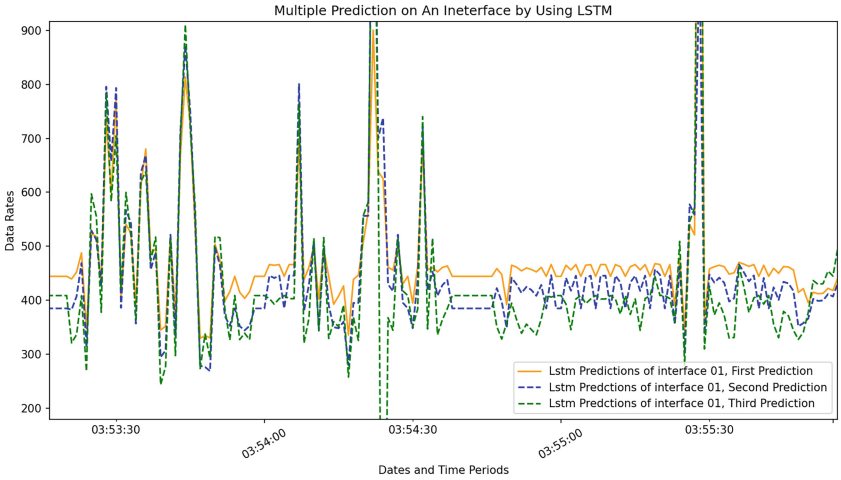

Fig. 16. Close view on Line graph of multiple prediction collected from interface-01 by using LSTM.

for the last rows of the prediction columns using the trained Random Forest model. After making predictions using the Random Forest Regression model, several evaluation metrics are computed to assess the model's performance. We use squared error as default evaluation metrics for this model. Below given figures shows the result of our work.

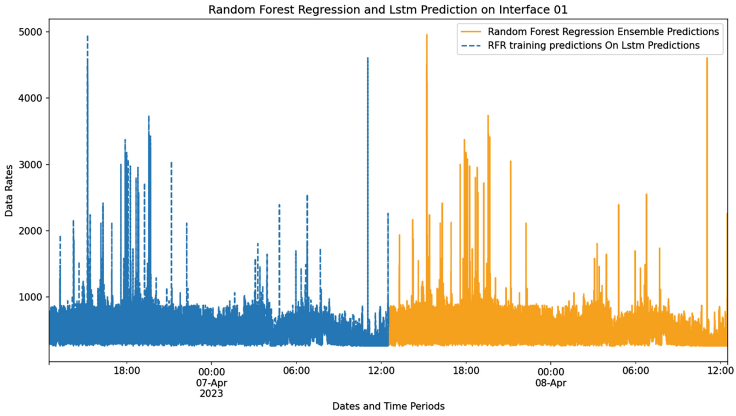

Fig. 17. LSTM and Random Foret Regression based generated predictions.

In Figs. 15 and 16 we have shown prediction results obtained by using LSTM. In this work we processed each interface traffic flow information dataset in LSTM. As a result of that processing multiple time implementation of LSTM on dataset, we obtained slightly

different future predictions results as clearly seen in the Figs. 17 and 18. For Random Forest Regression Ensemble, we have prepared new dataset which is previously observed predictions multiple times by using LSTM. In Random Forest Regression Ensemble our dataset containing multiple predictions with equal timestamp. Figures 17 and 18 shows prediction results after processing on Random Forest Regression Ensemble model. As we have discussed in above paras that Random Forest Regression Ensemble processes dataset by making tree and by taking mean of all data input.

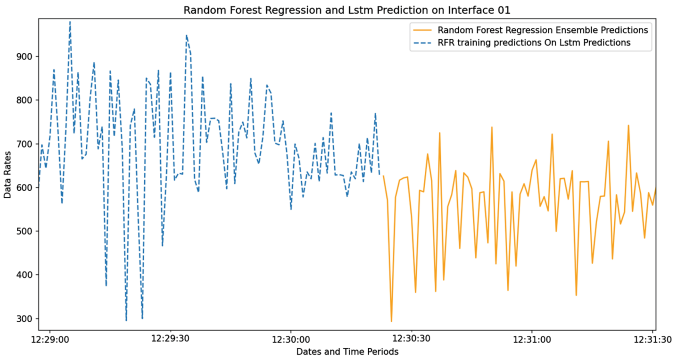

Fig. 18. Close view of LSTM and Random Foret Regression based generated predictions.

5 Conclusion

In conclusion, our work has taken the initial steps towards developing a comprehensive framework for Quality of Service (QoS) and Load balancing. We have successfully covered the prediction of data rates using LSTM in conjunction with the innovative utilization of Random Forest Regression Ensemble techniques. This optimization methodology, which enhances the accuracy of predictions, sets our work apart from previous research efforts. The groundwork laid here serves as the foundation for our future research endeavors in this field. In this study, we have employed two distinct Machine learning models for predicting future trends. The results underscore the significant role of Machine learning in managing network traffic, particularly in the context of emerging technologies. Our analysis clearly demonstrates that the integration of Machine learning with SDN, Fog Computing, RESTful API, and other cutting-edge technologies has a profound and far-reaching impact on the future landscape of networking technologies. Machine learning applications have the potential to revolutionize network traffic flow control. Through this research, we have meticulously examined various factors that warrant further investigation in future studies and research endeavors. Our work serves as a stepping stone towards a more efficient and adaptive network management system.

Acknowledgment. I would like to express my heartfelt gratitude for the invaluable support provided by “Research on Energy Hybrid Storage Optimization for Edge Computing” (2021FNA04016). Their steadfast commitment and invaluable contributions have significantly

enriched the outcomes of this study. This support has alleviated many challenges and enabled a comprehensive exploration of the subject. Their dedication to advancing knowledge and fostering innovation is evident in their multifaceted involvement, which transcends mere funding to encompass mentorship, access to cutting-edge resources, and intellectual exchange. Their generous support has been the cornerstone of this achievement, for which I am profoundly grateful.

References

1. Alghamdi, A., Paul, D., Sadgrove, E.: A RESTful northbound interface for applications in software defined networks. In: Proceedings of the 17th International Conference on Web Information Systems and Technologies (WEBIST 2021), pp. 453–459 (2021). ISBN: 978-989-758-536-4; ISSN: 2184-3252
2. Chu_, C.-Y., Xiy, K., Luoz, M., Jonathan Chao, H.: Congestion-aware single link failure recovery in hybrid SDN networks. In: 2015 IEEE conference on Computer Communications (INFOCOM), 9778-1-4799-8381-0/15/$31 ©2015 IEEE
3. Zakia, U., Ben Yedder, H.: Dynamic Load Balancing in SDN-Based Data Center Networks, 978-1-5386-3371-7/17/$31.00 ©2017 IEEE
4. Lon, H., Guo, M., Tang, F.: LABERIO: dynamic load-balanced routing in OpenFlow-enabled networks. In: 2013 IEEE 27th International Conference on Advanced Information Networking and Applications, 1550-445X/13 $26.00 © 2013 IEEE. https://doi.org/10.1109/AINA.2013.7
5. Tran∗, H.M., Le†, S.T., Nguyen†, S.V., Le, H.-D.: A web-based management system for software defined network controllers. In: 2019 International Conference on Advanced Computing and Applications (ACOMP), 978-1-7281-4723-9/19/$31.00 ©2019 IEEE. https://doi.org/10.1109/ACOMP.2019.00008
6. Rivera, S., Fei, Z., Griffioen, J.: RAPTOR: A REST API TranslaTOR for OpenFlow Controllers. In: 2016 IEEE INFOCOM International Workshop on Computer and Networking Experimental Research Using Testbeds, 978-1-4673-9955-5/16/$31.00 ©2016 IEEE
7. Chowdhury, S.R., Bari, Md. F., Ahmed, R., Boutaba, R.: PayLess: a low cost network monitoring, framework for software defined networks, 978-1-4799-0913-1/14/$31.00 c 2014 IEEE, 2014 IEEE Network Operations and Management Symposium (NOMS), 5-9 May 2014. https://doi.org/10.1109/NOMS20222.2014
8. Hua, T., et al.: SEAPP: a secure application management framework based on REST API access control in SDN-enabled cloud environment. J. Parallel Distrib. Comput. **147**, 108–123 (2021)
9. Zhou, W., Li, L., Luo, M., Chou, W.: REST API Design Patterns for SDN Northbound API, 2014 28th International Conference on Advanced Information Networking and Applications Workshops (2014)
10. Lin, S.-C., Wang, P., Akyildiz, F., Luo, M.: Towards optimal network planning forsoftware-defined networks. Trans. Mob. Comput. IEEE. https://doi.org/10.1109/TMC.2018.2815691
11. Agarwal, S., Kodialam, M., Lakshman, T.V.: Traffic engineering in software defined networks. In: 2013 Proceedings IEEE INFOCOM, 978-1-4673-5946-7/13/$31.00 ©2013 IEEE
12. Akyildiz, I.F., Lee, A., Wang, P., Luo, M., Chou, W.: A roadmap for traffic engineering in software defined networks. Comput. Netw. xxx (2014). https://doi.org/10.1016/j.comnet.2014.06.002, 1389-1286/_2014 Elsevier B.V. All rights reserved
13. Ramachandran, M., Archana, T., Deepika, V., Arjun Kumar, A., Sivalingam, K.M.: 5g network management system with machine learning based analytics. IEEE Access, Digital Object Identifier https://doi.org/10.1109/Access.2022.3190372

14. Latah, M., Toker, L.: Artificial intelligence enabled software defined networking: a comprehensive overview, © The Institution of Engineering and Technology 2018, IET Netw., **8**(2), 79–99 (2019)
15. Cisco Quality of Service Order of Operation. https://www.cisco.com/c/en/us/support/docs/quality-of-service-qos/qos-packet-marking/22141-qos-orderofop-3.pdf
16. Policing and Shaping Overview, QoS: Policing and Shaping Configuration Guide, Cisco IOS Release 15M&T. https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/qos_plcshp/configuration/xe-16/qos-plcshp-xe-16-book/qos-plcshp-oview.pdf
17. Distributing Bandwidth Between Queues, Americas Headquarters, Cisco Systems, Inc., 170 West Tasman Drive, San Jose, CA 95134-1706 USA © 2007 Cisco Systems, Inc. All rights reserved
18. Configuring Committed Access Rate, QoS: Classification Configuration Guide, Cisco IOS Release 15M&T. https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/qos_classn/configuration/15-s/qos-classn-15-s-book/qos-classn-car.pdf
19. Mirjalily, G., Akhavan Sigari, F., Saadat, R.: Best multiple spanning tree in metro Ethernet networks. In: 2009 Second International Conference on Computer and Electrical Engineering. 978-0-7695-3925-6/09 $26.00 © 2009 IEEE. https://doi.org/10.1109/ICCEE.2009.200
20. Hui, W., ZhongSheng, W.: Research on load balancing algorithm based on MSTP. In: Dynamic Network, 2012 International Conference on Industrial Control and Electronics Engineering, 978-0-7695-4792-3/12 $26.00 © 2012 IEEE. https://doi.org/10.1109/ICICEE.2012.94
21. de Sousa, A.F.: Improving load balance and resilience of Ethernet carrier networks with IEEE 802.1S multiple spanning tree protocol. In: Proceedings of the International Conference on Networking, International Conference. 0-7695-2552-0/06 $20.00 © 2006
22. Li, L., Chuo, W., Luo, M.: Design patterns and extensibility of REST API for networking applications. IEEE Trans. Netw. Serv. Manag. https://doi.org/10.1109/TNSM.2016.2516946