## TRAFFIC AWARE OPTIMAL ROUTING IN SDN BY PREDICTING EDGE WEIGHTS USING NEURAL NETWORK

## Project Report

***Submitted by***

|  |  |
| --- | --- |
| Manoj S | 2019506048 |
| Aravinth Kumar A M | 2019506012 |
| Vimal V | 2019506114 |

***Under the supervision of***

Dr. S Umamaheswari

*In partial fulfilment for the award of the degree of*

# BACHELOR OF TECHNOLOGY

*in*

**INFORMATION TECHNOLOGY**



# DEPARTMENT OF INFORMATION TECHNOLOGY MADRAS INSTITUTE OF TECHNOLOGY CAMPUS ANNA UNIVERSITY, CHENNAI – 600044

ANNA UNIVERSITY: CHENNAI 600 025

# BONAFIDE CERTIFICATE

Certified that this mini-project report titled “**TRAFFIC AWARE OPTIMAL ROUTING IN SDN BY PREDICTING EDGE WEIGHTS USING NEURAL NETWORK**” is the bonafide work of Manoj S (2019506048), Aravinth Kumar A M (2019506012) and Vimal V (2019506114) who carried out the project under my supervision.

|  |  |
| --- | --- |
| **Signature** | **Signature** |
| **Dr. M. R. Sumalatha** | **Dr. S. Umamaheswari** |
| **HEAD OF THE DEPARTMENT** | **SUPERVISOR** |
| Professor | Associate Professor |
| Department of Information Technology | Department of Information Technology |
| MIT Campus, Anna University  Chennai – 600044 | MIT Campus, Anna University  Chennai – 600044 |

# ACKNOWLEDGEMENT

It is essential to mention the names of the people, whose guidance and encouragement made us accomplish this project.

We express our gratitude and sincere thanks to our respected Dean of MIT Campus, **Dr. J. Prakash**, for providing excellent computing facilities throughout the project.

Our sincere thanks to **Dr. M. R. Sumalatha**, Head of the Department of Information Technology, MIT Campus for catering all our needs giving out limitless support throughout the project phase.

Our sincere thanks to **Dr. Radha Senthilkumar**, Associate Professor, MIT Campus for invaluable feedback in reviews

We express our thankfulness to our project supervisor **Dr. S. Umamaheswari** Teaching fellow, Department of Information Technology, MIT Campus, for providing invaluable support and assistance with encouragement which aided to complete this project.

|  |  |
| --- | --- |
| **MANOJ S** | **2016506048** |
| **ARAVINTH KUMAR A M** | **2016506012** |
| **VIMAL V** | **2016506114** |

**ABSTRACT**

Packet-switching is one of the significant methods for transmitting the packets over the traditional network. Due to the quick development of networking technologies and network devices, the packet-switched systems frequently experience a rapid growth of the traffic over the traditional network which places a large and unbalanced burden on the routers. Henceforth, we move towards a traffic-based routing method while transmitting the packets to their destination over the network. This traffic-based routing enables the network devices (routers, switches etc.) to be utilized to their fullest. Any network's throughput and link-to-link delay can be considerably increased and decreased respectively by comprehending traffic and changing pathways. Traditional routing algorithms operate in the data plane, making it impossible to gather sufficient network statistics. Software-defined networks are used in data centre networks to analyse and handle traffic. This enables the network administrator to monitor, maintain and install routing paths in the data layer nodes. Even if SDN decouples the control plane from the data plane, a scalable and traffic-aware algorithm is required allowing to make efficient routing decisions. This project aims to provide optimal routing in SDN by defining protocols to sample network performance metrics from the data layer at different time sequence based on which the future traffic would be predicted using GRU (Gated Rectilinear Unit). The predicted traffic would be used in determining the edge weights between links in real time which would previously been impossible to obtain. when an outbound packet is received it can be routed optimally by choosing a path with less traffic which improve the QoS (Quality of Service) of routing the packets.

**TABLE OF CONTENTS**

**CHAPTER NO TITLE PAGENO**

**ABSTRACT 4**

**LIST OF FIGURES 7**

**LIST OF TABLES 8**

**LIST OF ABBREVIATIONS 9**

1 **INTRODUCTION**

1.1 Overview 10

1.2 Research Challenges 11

1.3 Problem Statement 11

1.4 Objective 12

1.5 Scope of the project 12

1.6 Organization of Report 13

2 **LITERATURE SURVEY**

2.1 Review of the Existing Work     14

2.2 Research Gap 15

2.3 Summary 16

3 **SOFTWARE DEFINED NETWORK**

3.1 SDN and its architecture 17

3.2 OpenFlow protocol 18

3.3 Network Performance Parameters 19

4 **PROPOSED WORK**

4.1 System Architecture 20

5 **IMPLEMENTATION**

5.1 Send init packet module 22

5.2 Neural Network module 24

5.3 Metric Extractor module 30

5.4 Path generation module 32

6 **VALIDATION**

6.1 Simulation Parameter 33

6.2 Neural Network Model 34

6.3 Shortest Path Implementation 38

7 **TOOLS AND TECHNIQUES**

7.1 GO language 39

7.2 ONOS Controller 39

7.3 Mininet Simulator 39

7.4 Languages Used 39

8 **CONCLUSION AND FUTURE WORK**

8.1 Conclusion 41

8.2 Future Work 41

9 **APPENDIX** 42

10 **REFERENCES** 44

**LIST OF FIGURES**

**Figure No Title Page no**

3.1 SDN Architecture         17

4.1 System Architecture 21

5.1 Send init packet module 22

5.2 Neural Network Architecture 26

5.3 LR vs Epoch graph in LR Scheduler 28

5.4 Comparison of LR based and non-LR based 28

Model

5.5 Entropy normalisation of predicted weights 30

5.6 Unnormalized metrics for one iteration 31

5.7 Normalized metrics for one iteration 31

6.1 Start Topology 33

6.2 Train Loss with initial tuning 35

6.3 Validation Loss with initial tuning 35

6.4 Train Loss fine tunned 36

6.5 Validation Loss fine tunned 36

6.6 Training cosine similarity 37

6.7 Validation cosine similarity 37

**LIST OF TABLES**

**Table no Title Page no**

6.1 Simulation environment           34

6.2 Initial hyperparameter tuning 34

6.3 Hyperparameter fine tuning         36

6.4 Comparison between predicted weights and equal 38

weights with Dijkstra’s algorithm

**LIST OF ABBREVIATIONS**

**NOTATION DESCRIPTION**

SDN Software Defined Networking

PLR Packet Loss Rate

API Application Programming Interface

REST Representational State Transfer

SNMP Simple Network Management Protocol

NetConf Network Configuration Protocol

SDVN VANET Based SDN

VANET Vehicular Ad Hoc Networks

**CHAPTER 1**

**INTRODUCTION**

* 1. **OVERVIEW:**

Traditional Network uses fixed and dedicated hardware devices such as routers and switches to control the traffic that occurs in the network. Because of its inability to scale large number of users and network security and performance to large number of users, they are in a major concern. Software Defined Network (SDN) helps in scaling efficiently and the main difference between the traditional network and a Software Defined Network (SDN) is that, the control layer and the data layer are decoupled in the SDN. In traditional network forwarding and routing are carried out in each hop, but in SDN forwarding is done in the data layer by the Forwarding Element (which is mostly a switch) and routing is done by a centralized controller. This allows the network administrator to program and control the entire network from a single view. In a traditional network there is no option to program, it is purely hardware based. In SDN we can control the network, change configuration, provision resources, and increase network capacity. With increased visibility and ability to define secure pathways SDN is more secure than traditional networks.

The goal of this project is to route packets efficiently by analysing the traffic in the SDN network and predicting the optimal path for routing the packet based on the network metrics obtained. Significantly based on the obtained parameters the network metrics for the future traffic can be predicted and subsequently the packets can be routed in an optimal path based on the network traffic in each path.

* 1. **RESEARCH CHALLENGES:**

There are several research challenges that are associated with analysing the network traffic and routing the packets efficiently.

**Obtaining the network performance parameters:** One of the main challenges in analysing the network traffic is obtaining the underlying network performance parameters. Network parameters are essential for analysing the network traffic in any underlying network. The network performance parameters such as delay, jitter, bandwidth, Packet Loss Rate (PLR) and queue utilization.

**Predicting the future traffic:** Another important research challenge is predicting the network performance parameters for predicting the future traffic to route the packets efficiently. This may involve training a neural network model for predicting the future network performance parameters to analyse traffic with the past network performance parameters

* 1. **PROBLEM STATEMENT**

Traffic based routing enables the network devices to be used to their fullest. By understanding traffic and altering paths we can significantly increase the throughput and reduce the link-to-link delay in any network. Traditional routing algorithms operate in the data plane and enough statistics about the network cannot be obtained. To handle traffic in data centre networks software defined networks are deployed, this allows the network administrator to monitor, maintain and install routing paths in the nodes of the data layer. Although SDN decouples the control plane from data plane allowing to make efficient routing decisions, a scalable and traffic aware algorithm is needed. Another problem is link failure in networks that can cause significant impact in the network throughput. The algorithm needs to efficiently detect network failures and install alternative paths and reduce the recovery time for normal traffic flow.

* 1. **OBJECTIVE**

The work proposes a technique to define protocols to obtain real time network performance metrics such as the delay, jitter and bandwidth from the host application. With the predicted nature of the traffic, the link-to-link cost is determined and is used by the routing scheme to install paths or flows among the networking devices (routers or switches) in the data plane using ONOS controller.

* 1. **SCOPE OF THE PROJECT:**

**In datacentres:**

Efficient routing algorithms can optimize the flow of packets in the datacentres.

**WAN and VANETs:**

As SDN enables decoupling of control plane and the data plane in SDVN (VANET Based SDN) provides,

* A network state and a networking intelligence that is conceptually centralised.
* An abstraction for the underlying network architecture for VANET (Vehicular Ad Hoc Network) applications.

**Sensor Networks:**

The routing algorithms proposed can optimize the communications between the IOT devices.

* 1. **ORGANIZATION OF REPORT:**

Chapter 1 gives a brief introduction about the differences between the traditional network and the SDN network in routing the data packets over the network. Chapter 2 analyses and gets various notions from international conference and journal papers. Chapter 3 describes Software Defined Network (SDN) and its architecture and some network parameters. Chapter 4 describes about the overall architecture of the system implemented. Chapter5 presents the implementation of each module in the proposed system. Chapter 6 gives a comparative analysis of this work and Dijkstra’s. Chapter 7 highlights the technologies used for this work. Finally, Chapter 8 gives a conclusion followed by the reference section which is stacked where details of various papers related to the project are added.

**CHAPTER 2**

**LITERATURE SURVEY**

**2.1 REVIEW OF THE EXISTING WORK:**

The work proposed in [1] involves obtaining metrics to evaluate network performance and assign weights to each parameter based on its entropy. The property of every attribute is emphasized and would help in decision making by using entropy weighting [14]. After extracting parameters, a decision matrix is constructed based on the requested path which is used to choose the best route. On analysis with varying topologies the algorithm performed better than shortest path or greedy algorithm in terms of AUC (area under curve) of network performance metrics. An efficient routing scheme to handle load balancing requirements is proposed in [2]. Here parameters like CPU resources, length of queue buffer and centrality of nodes in the network are extracted and PCA normalization is applied and used to predict queue utilization (QU). By using QU for their algorithm, they were able to significantly reduce delay and increase traffic throughput in comparison with bellman-ford algorithm.

To adapt complex network Hauang, L [3] proposed GRU (Gated Recurrent Unit) based traffic prediction and Duelling DQN (reinforcement learning) to generate the forwarding paths. With GRU the model was able to find hidden and unknown network traffic status change and the reinforcement algorithm was able to converge to optimal path in very a smaller number of iterations and provided better throughput than OSPF and Dijkstra. While the above algorithms were able to make decisions based on traffic data the reaction on link failure is not studied. A novel efficient EFSUTE algorithm was proposed in [4] which identifies link failure using Bidirectional Flow Detection (BFD) protocol and proactively updates an alternative path. It used the SRLG (Shared Resource Link Group) algorithm to identify disjoint paths by using delay, link utilization etc., as cost. This method decreased the recovery time on different topologies.

**2.2 RESEARCH GAP:**

|  |  |  |
| --- | --- | --- |
| **Paper** | **Pros** | **Cons** |
| Automatic Software Defined Network (SDN) Performance Management Using TOPSIS Decision-Making Algorithm - springer journal of Grid Computing (2021) | TOPSIS algorithm is implemented which ranks adjacent routes based on network parameters with entropy-based weight assigning | Delay sensitive packets were tested but not prioritized in routing |
| EFSUTE: a novel efficient and survivable traffic engineering  for software defined networks - springer journal of Reliable Intelligent Environments (2020) | Proactively mitigates network node failure by using SRLG algorithm to identify 2 paths which reduce bandwidth costs | The average traffic flow is considered but node related traffic parameters is not considered for optimization |
| Intelligent routing method based on Dueling DQN reinforcement  learning and network traffic state prediction in SDN - springer journal of Wireless Networks (2022) | Estimated future network parameters by using time series data with GRU network and chose the optimal path by fast converging reinforcement learning algorithm with predicted network parameters | Node failure scenarios were not discussed |
| Machine Learning Aided Load Balance Routing Scheme Considering Queue Utilization - IEEE Transactions on Vehicular Technology (2019) | DL algorithm was built to predict queue utilization, the next hop was determined using QU to optimize routing | Enough stress was not given to route packages based on the destination |
| Fast and efficient algorithm for delay-sensitive QoS provisioning  in SDN networks - Wireless Networks Springer nature (2022) | Proposed a lagrangian optimization-based algorithm to give the optimal path by reducing the number of internal Dijkstra calls by 15% thereby computing path in shorter time and hence reducing delay | Scenarios for varying traffic load are not handled and tested. |

**SUMMARY:**

Although several techniques exist to obtain relevant traffic information from the data layer, different problems which could arise on sampling data is not discussed. Further traffic obtained is not validated with the current actual traffic.

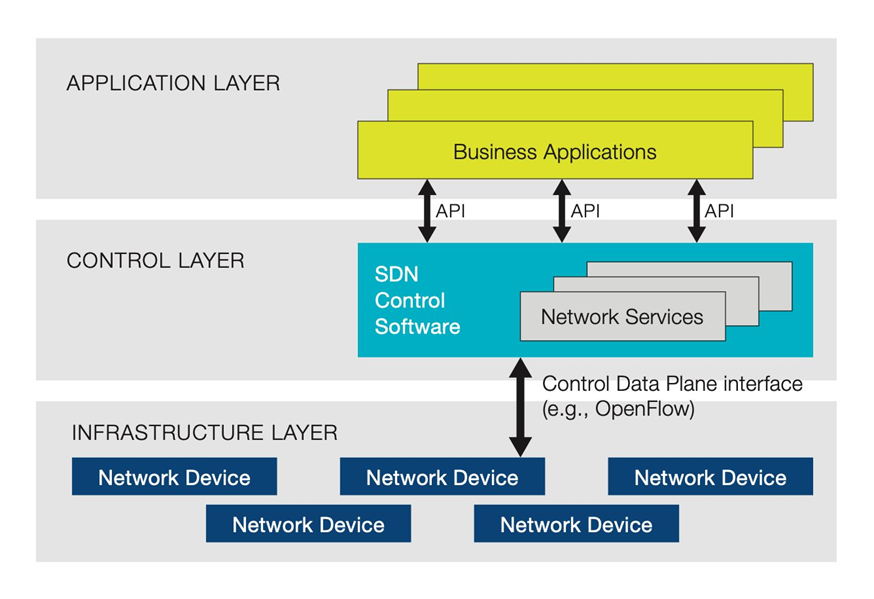
**CHAPTER 3**

**SOFTWARE DEFINED NETWORK**

**3.1 SDN AND ITS ARCHITECTURE:**

Software-Defined Networking (SDN) is a network architectural approach that enables us to build programmable and efficient networks. It uses controllers or APIs to communicate with the hardware infrastructure. SDN provides the following features:

* Increased Control
* Customizable network infrastructure
* Security



**Fig. 3.1. SDN Architecture**

In SDN the data layer and control layer are decoupled as visualized in Fig 3.1. In traditional network forwarding and routing are carried out in each hop, but in SDN forwarding is done in the data layer by the Forwarding Element (which is mostly a switch) and routing is done by a centralized controller. This allows the network administrator to program and control the entire network from a single view.

The architecture of SDN can be divided into three parts:

* **Application Layer:** This layer enables interaction with users to configure parameters of the network or monitor the network. The application layer relates to the control plane with the Northbound API which is usually built using REST / SNMP / Netconf
* **Control Layer:** In this layer decisions about where to route the packet can be made. It interacts directly with the data plane with the Southbound API which is mostly OpenFlow
* **Data Layer:** Network devices receive information from controllers and update their routing table. This is where actual packet transfer happens

In a traditional network there is no option to program, it is purely hardware based. In SDN we can control the network, change configuration, provision resources, and increase network capacity. With increased visibility and ability to define secure pathways SDN is more secure than traditional networks.

**3.2 OPENFLOW PROTOCOL:**

It enables communication between controllers and network devices. It works on TCP, OF connection is established after a 3-way handshake between controller and network device. The switch’s MAC table which stores the host's network address is now called Flow tables. Different types of SDN communication:

* **Open SDN:** There is no intelligence in switch, the controller determines where to route the packets and switch updates the forwarding table based on the information provided by controllers
* **SDN with API:** The switches have little intelligence and can implement logic on choosing what kind of packets should be sent to controllers.
* **SDN overlay model:** A virtual network is run on top of existing hardware creating dynamic tunnels to different on-premise and remote data centres.
* **Hybrid SDN:** It combines SDN with traditional network protocols in one environment to support different functions on the network.

**3.2 NETWORK PERFORMANCE PARAMETERS:**

Delay: It is the average time interval from source to destination for each successfully delivered packet [4]. Jitter: It is measured by calculating the average time difference between each packet sequence. [11]. Package loss rate: It is the total number of data packets which have not been delivered at destination to the total number of packets streamed at source [4]. Bandwidth: The maximum amount of data transmitted over an internet connection in each amount of time. Queue Utilization: Amount of buffer used in switch at given instance of time [2]. Recovery time is the time elapsed between reception of first packet after link failure and the reception of last packet before link failure [13]

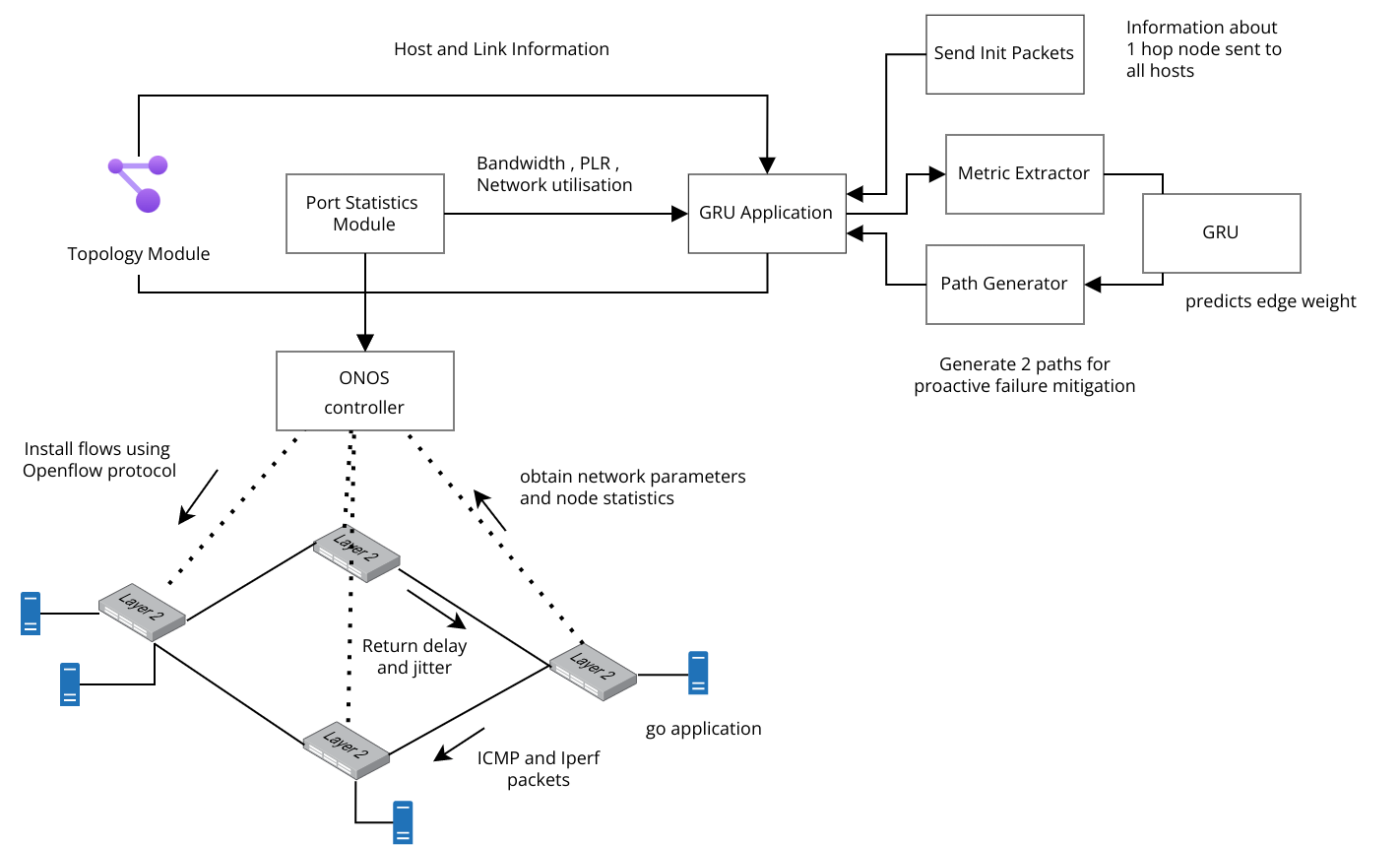
**CHAPTER 4**

**PROPOSED WORK**

**4.1 SYSTEM ARCHITECTURE:**

The ONOS controller is connected to a Mininet with suitable topology built. Instead of building an application which instructs the controller through northbound API the GRU Application is written alongside with controller core functions. By this technique the application would be able to access modules of the controller. For instance, the topology information of the underlying network is monitored by a topology module which sends LLDP (link layer discovery protocol) [10] packets and obtains the required information. On building an application alongside the core of ONOS we have access to network topology through function calls from the mentioned module. The mentioned application is built using java language. The main component of the application is the packet processor which provides an interface for accessing the packets directed to the controller and the packets whose route is not determined at the data layer.

The entire architecture can be divided into 4 major modules as shown in Fig 3.2: send Init packets, metric extractor, GRU layer and path generator. All these modules directly interact with the controller and the overall objective is to understand the network traffic and install flows to optimize different network performance parameters.

****

**Fig. 4.1 System Architecture**

**Algorithm 4.1: Routing in proposed algorithm**

**Step1:** The controller application sends a packet containing nearby host information.

**Step2:** Host application captures the packet and sends delay and jitter for requested nearby hosts.

**Step3:** Obtained parameters is normalized using average value.

**Step4:** The normalized values are then given as input into GRU Neural network.

**Step5:** The Normalized value is then converted to actual metric and each iteration is Normalized using Entropy normalization.

**Step6:** With the normalized values edge weights are determined for every link.

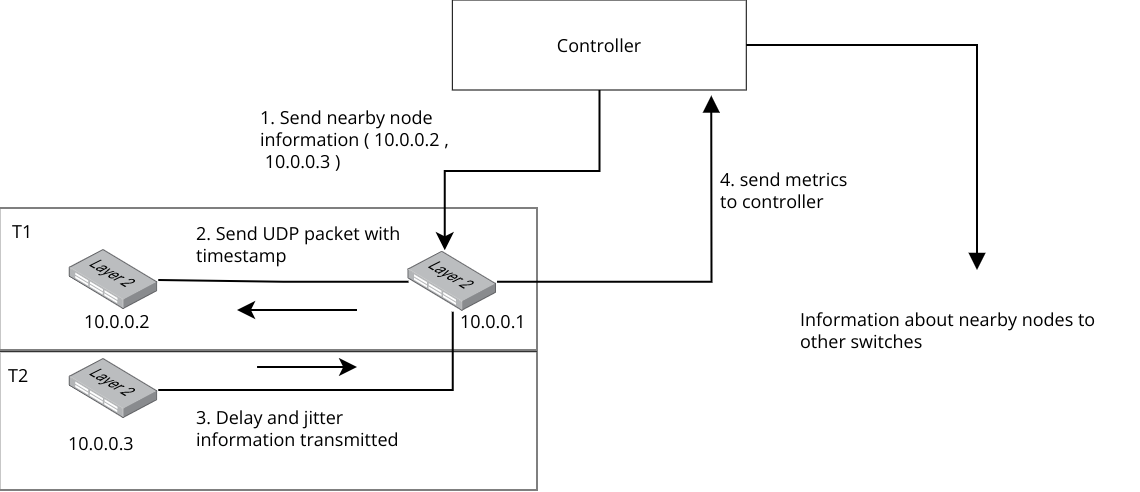
**Step7:** Shortest path algorithm is applied with the weights and path is installed in the data plane until it expires.

**CHAPTER 5**

**IMPLEMENTATION**

**5.1 SEND INIT PACKET MODULE**

Understanding the traffic flow is necessary to generate optimal routing algorithms. The controller does not store information about the link-to-link delay of packets or any other performance parameter. The only way to measure is to monitor from the host application. An example of how metrics are received from data layer is show in Fig 5.1.

****

**Fig. 5.1 Send Init packet Module**

One approach for a host application to send delay and jitter information to the controller is to generate broadcast ping requests. ICMPv4 allows broadcast ping messages, but the problem with this approach is that broadcast generates and receives unnecessary packages. That is information about 2 hop hosts is not necessary. Since iperf command is built on top of TCP when parallel connections are established the working is thwarted. So, to handle this an application built using go language is installed in host which frames its own UDP packet that contains the timestamp at which the packet is framed. The timestamp is of the format RFC3339Nano. This timestamp is obtained from the receiving end at another host application and delay is calculated.

**Algorithm 5.1: Host application to receive and send metric information**

**Input:** packets from the controller: C­­i, packet from nearby host: Pi, number of packets to transfer: N

**Output:** packet to another host: Po, packet to Controller Co

1: Pi ← Server listening to another host (Implemented as separate thread)

2: **While** true

3: Ci ← Server listening to Controller

4: A ← get the list of nearby hosts from Ci

5: W ← wait group

6: **For** Hs ← A

7: Add 1 to W

8: Send N packets to Hs containing current timestamp. After sending N packets remove 1 from W. This is done in separate thread.

9: Wait for all items in W

10: **End For**

11: **End While**

12: Initialize Mp: Map hs to array of delay

13: **Process Packet (Pi)**

14: Ts, Hs ← Pi

15: If first packet start timer of 2s for Hs (Separate thread)

16: D ← Tcurrent – Ts

17: Add D to Mp[Hs]

18: If all N packets received Send for Hs → Send Packet Out (Hs) (Contains delay and jitter)

19: **Finish Process Packet**

A metric server is started in every application as a separate thread, the purpose of this server is to listen to UDP packets sent by a nearby host containing the timestamp. A specific host sends 10 packets at a time for one iteration each bearing its own timestamp. As a packet is received the time stamp is extracted and the delay is calculated. When the first packet of a specific host is received a thread which runs for 2 seconds is started, at the end of the thread a go channel is updated to set timeout. If all the other 9 packets are received within the next 2 seconds the same channel is set to completed state. This allows the application to continue with average delay calculation instead of waiting for all 10 packets in case of packet loss. With the delay of 10 packets the jitter is calculated.

The host application needs to send time stamped packets to multiple hosts, to handle it efficiently the packets are sent simultaneously to different hosts. But while sending packets simultaneously the packets of one iteration should not collide with that of next. So, a wait group is created which blocks until all the packets for one iteration are sent. Sending packets simultaneously also ensures the metrics are calculated at the same time between different hosts.

After the metrics are received an UDP packet containing the metrics is sent to the controller The above process is repeated every 5 seconds so the controller could get real time traffic updates. The 5 seconds set here is a hyperparameter which can be tuned. Bandwidth of every link is sampled directly from the controller using the port statistics module. The packet received by controller is extracted and relevant information is stored in map which is stored in a CSV for training the neural network model.

**5.2 NUERAL NETWORK MODULE**

The objective of this module is to generate the future metrics given a sequence of metrics which will be used to calculate the edge weights. A neural network is built using pytorch library in python language.

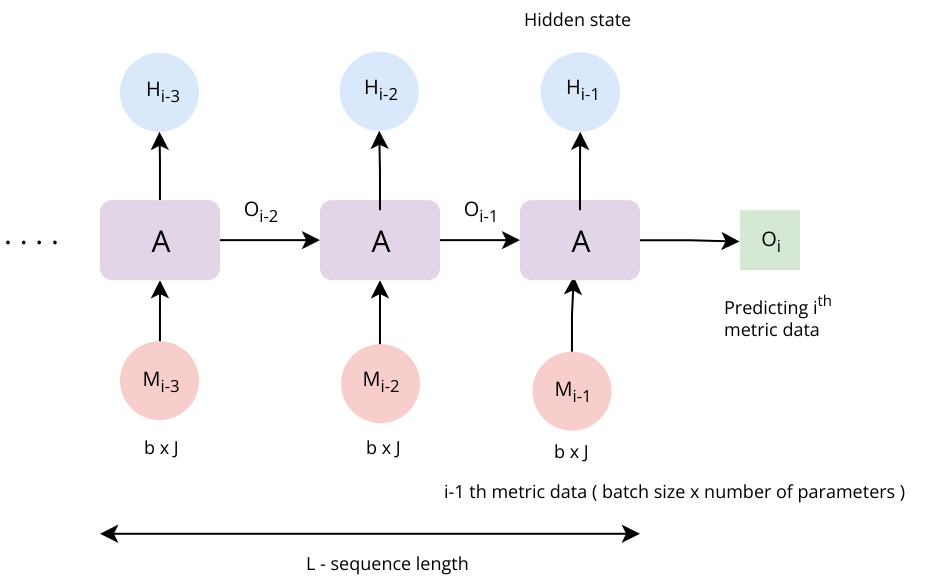
**5.2.1 DATA DESCRIPTION AND NORMALIZATION**

Each data point as previously explained represents the network parameters per iteration per link. The simulation with star topology was run for 10 thousand seconds in star topology which is nearly 2 hours 45 minutes, this generated around 20 thousand data points. The unnormalized data was used because entropy normalization is specific to an iteration, so the parameter of one iteration would not be relatable leading to poor performance of the model.

The neural network performs better to data points less than 1. So, a different normalization is proposed: the average of each parameter is taken and a suitable value relating to it is used to normalize the data points.

**5.2.2 GRU AND FULLY CONNECTED LAYER**

The sequence of metric data generated is used as input to the RNN to predict the future metrics. It takes the previous L sequence as input and predicts the L+1th sequence. The sequence is obtained from the iteration id. GRU has fewer parameters than LSTM. Fewer the parameters lessen the chances of the model getting overfitted. The GRU network supports two gates, the reset and update gates. In the Fig 5.2 Mi stands for the metric generated at ith time sequence. It is a matrix of size b x J where b is the batch size, here taken as the number of links in the topology and J stands for the number of network performance parameters. Oi is used for calculating the link weights.

****

**Fig. 5.2 Neural Network architecture**

The Unidirectional GRU module is initialized with 1024 hidden parameters and 1 hidden layer. The final output contains 3 values denoting the number of parameters. But on combining the hidden sequence the output is of size (N, L, D\*H) where N is the batch size, L is the sequence length D representing 1 in case of unidirectional GRU and H is the number of output parameters which is 3. It is then converted into a tensor of size (N, L\*D\*H).

The converted tensor is passed into a fully connected neural network which converts the tensor into size (N: number of network parameters) which would then be used against the future sequence with the loss function.

**5.2.3 TRAINING GRU MODEL**

The following are the hyperparameters Used:

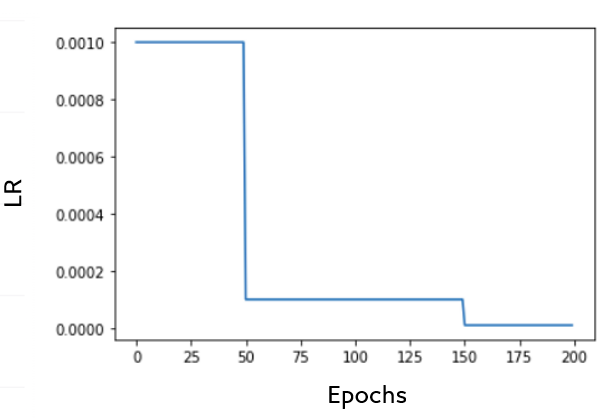
* L - sequence length - the number of previous iterations to be considered for training.
* Number of epochs
* Learning Rate
* Number of RNN layers
* Dropout percent
* Number of hidden layers in RNN

Different values for each of the hyperparameters is tested and the model is trained in a kaggle GPU P100 notebook. All the operations were done in floating point (32 bits), the batch size is the number of links as already mentioned. 80% of the available data is used for training and the remaining 20% is used for validation. The model is made to run for 250 epochs. It took around 7 and half hours to train the whole model with GPU.

Mean squared error (MSE) is used as training loss. ADAM (adaptive moment estimation) is used as the optimizer for the model. Learning rate scheduler is used for fixing the learning rate with initial LR value of 0.001.

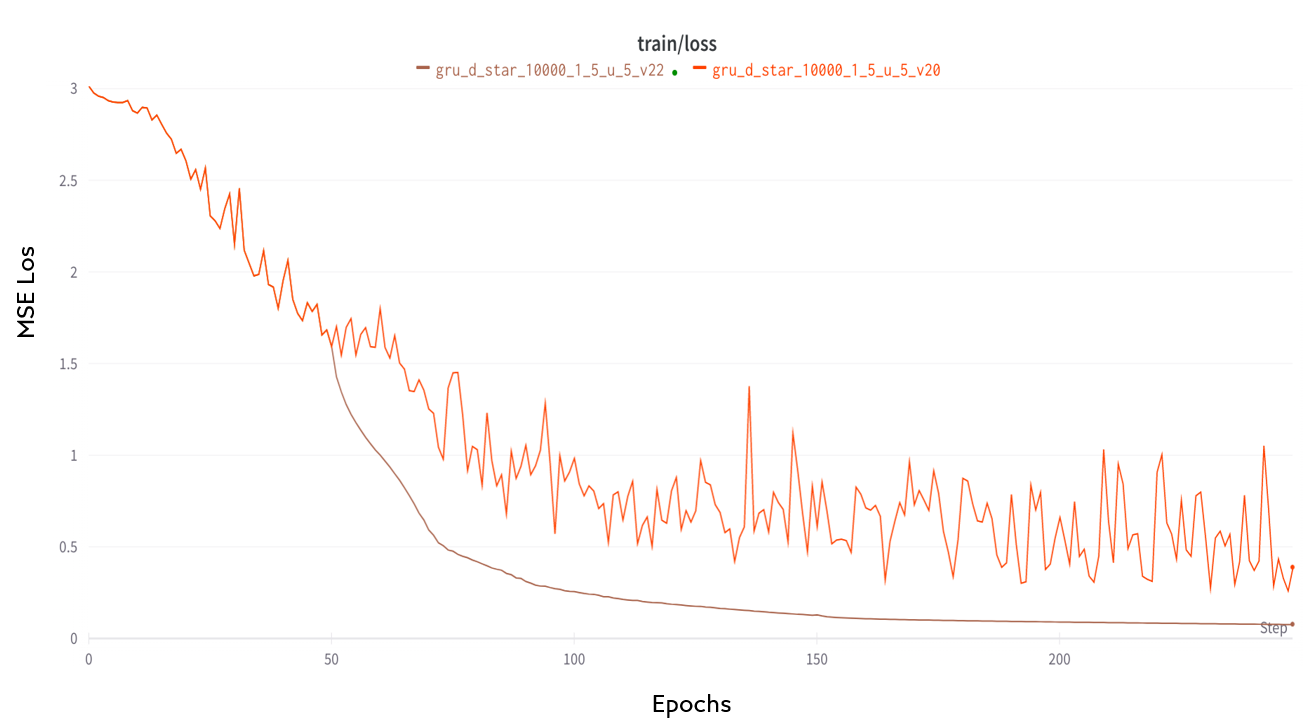
**5.2.4 LEARNING RATE (LR) SCHEDULER**

On running the model with many epochs, the training loss fluctuates as shown in the figure (peach colored curve). To tackle this issue an LR scheduler is used. The LR scheduler reduces the learning rate after a certain number of epochs which provides a fine-tuning effect. This is better than using a low learning rate because on low LR the model would never converge. MultistepLR is used which reduces the LR by a factor of gamma after a certain number of epochs as shown in the Fig 5.3.



**Fig. 5.3 LR vs Epochs graphs in LR Scheduler**

The same hyperparameters are used for both LR scheduler-based model and Non LR scheduler-based model and shown in the Fig 5.4.



**Fig. 5.4 Comparison of LR-Based and Non-LR-Based Model**

It can be seen from the graph that after 50 epochs the LR based model (brown) performs drastically better than the model without LR (orange). Train MSE loss has gone down from 0.389 to 0.079.

**5.2.5 DEPLOYING THE MODEL**

The trained model is converted to an ONNX file (Open Neural network exchange) which allows it to be called from java. The ONNX runtime is used in the java end to load the model and feed in the inputs. The parameters are represented as a map of floating-point arrays. Each point in the map represents a link in the topology. The metrics are normalized as mentioned in the metric extractor module. A linked list of size 5 is used to store the data points of the previous sequence. When the predict function is called from the timer function the linked list buffer is converted into a single float of dimension (N, L, number of parameters) where N is the number of links, L is sequence length.

The data is then normalized and fed into the loaded ONNX model. The output is of size (N, number of parameters) the weights are then calculated for each link and stored as a map.

**Algorithm 5.2: Predicting future metrics using GRU model**

**Input:** GRU Model: M, Metric obtained every 5 seconds: Li, Number of Sequence to consider: Ns, Number of Links: Ni, Number of metric parameters Np

**Output:** Predicted Metrics Pi

1: Li ← Timer Function called every 5 seconds returns the current capture of metrics

2: Initialize Linked List B

3: **While** True

4: **If** length(B) < Ns **then**

5: Push Back (B, Li)

6: **else**

7: Pop Front (L­i)

8: F ← Convert B into 3-dimensional array of size (Ni, Ns, Np)

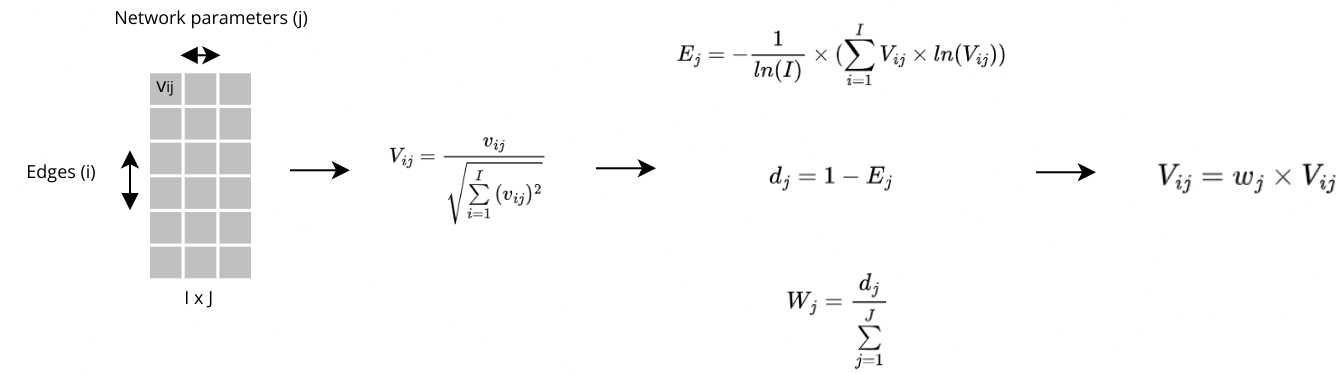
9: Pi ← M(F)

10: **endif**

11: **return** P­i

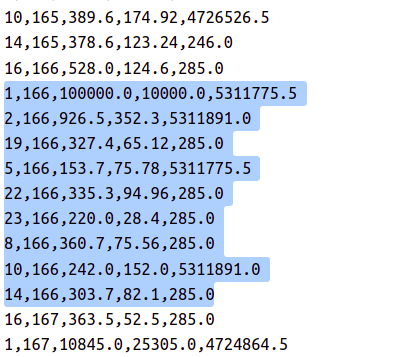
**5.3 METRIC EXTRACTOR MODULE**

A packet processor is implemented in the controller application. The objective of the packet processor is to analyse the packets received to the controller which may be control information, metrics sent by the host application or packet which is not yet routed.

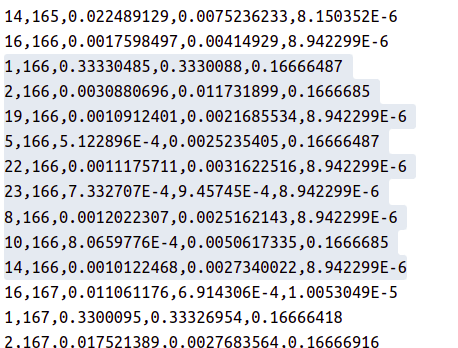
****

**Fig. 5.5 Entropy Normalization of predicted weights**

The metrics obtained from the hosts are represented in the form of a matrix as shown in the Fig 5.5. The column of the matrix represents a specific network parameter here in the case delay, jitter, and bandwidth. The row represents a particular link. The matrix is normalized parameter wise using vector normalization. The entropy (E) represents the varying nature of the given parameter. Deviation (d) is complementary to entropy. More deviation the more the network parameter influences network performance. Weights (W) of each parameter is the amount of deviation contributed by each parameter. Each value (V) is then multiplied by its corresponding weight. On overall multiplying it with weight calculated gives values corresponding to the diversity of each parameter. The Fig 5.6 and Fig 5.7 shows the normalized and unnormalized network performance parameters for delay, jitter, and bandwidth for every 5 seconds interval with help of timer function in java application.

****

**Fig. 5.6 Un Normalized metrics for one iteration**

****

**Fig. 5.7 Normalized metrics for one iteration**

Each line in both the above images represents the parameters per iteration per link. The first value represents the link, this is denoted as a single value i.e19 represents link between (19//5 and 19%5), (5,3). The next value represents the iteration id and the last three values correspond to delay, jitter, and bandwidth. The figure shows both normalized and unnormalized values. Then these values are written csv file for training the GRU neural network.

**5.4 PATH GENERATION MODULE**

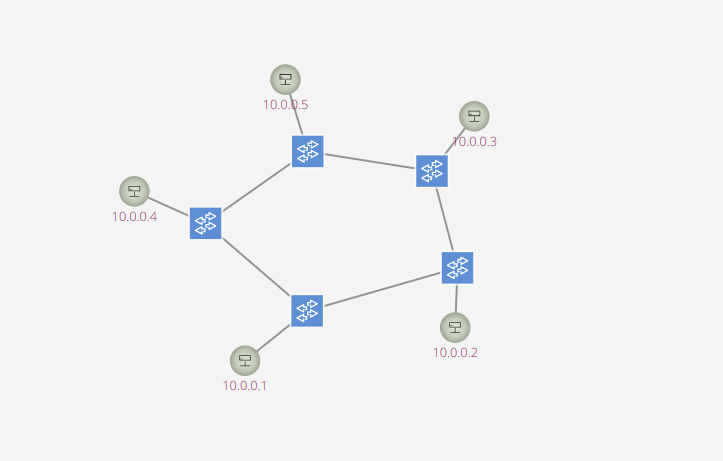
With the predicted network performance parameters, a shortest path routing algorithm is used. After generating the shortest which is represented as array ONOS link data structure it is and then the path is validated that is all the links are checked. This happens each time a packet is received whose route cannot be determined. The path is then installed in the specified routers each of the route has a timeout period. After the path gets expired the same process happens again and a different path is installed based on the traffic on the data layer.

**CHAPTER 6**

**VALIDATION**

**6.1 SIMULATION ENVIRONMENT**

The above implementation is done in star topology as shown in Fig 6.1. For better validation of the result, New York city centre topology, Mesh, partial Mesh would be considered and the resulting network performance parameters would be compared with Dijkstra.

****

**Fig. 6.1 Star Topology**

CISCO defines network traffic to be of two types: elephant flows which are packets of large size and mice flows (small size). Based on this random traffic both TCP and UDP were generated using the iperf command.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Name** | **number of FE** | **max bw(mbps)** | **min bw(mbps)** | **min delay(ms)** | **max delay(ms)** | **time (mins)** | **mice flows** | **elephant flows** |
| star1 | 5 | 100 | 10 | 10 | 120 | 15 | 5 | 1 |
| star2 | 5 | 100 | 10 | 10 | 120 | 15 | 5 | 3 |

**Table. 6.1 Simulation Conditions**

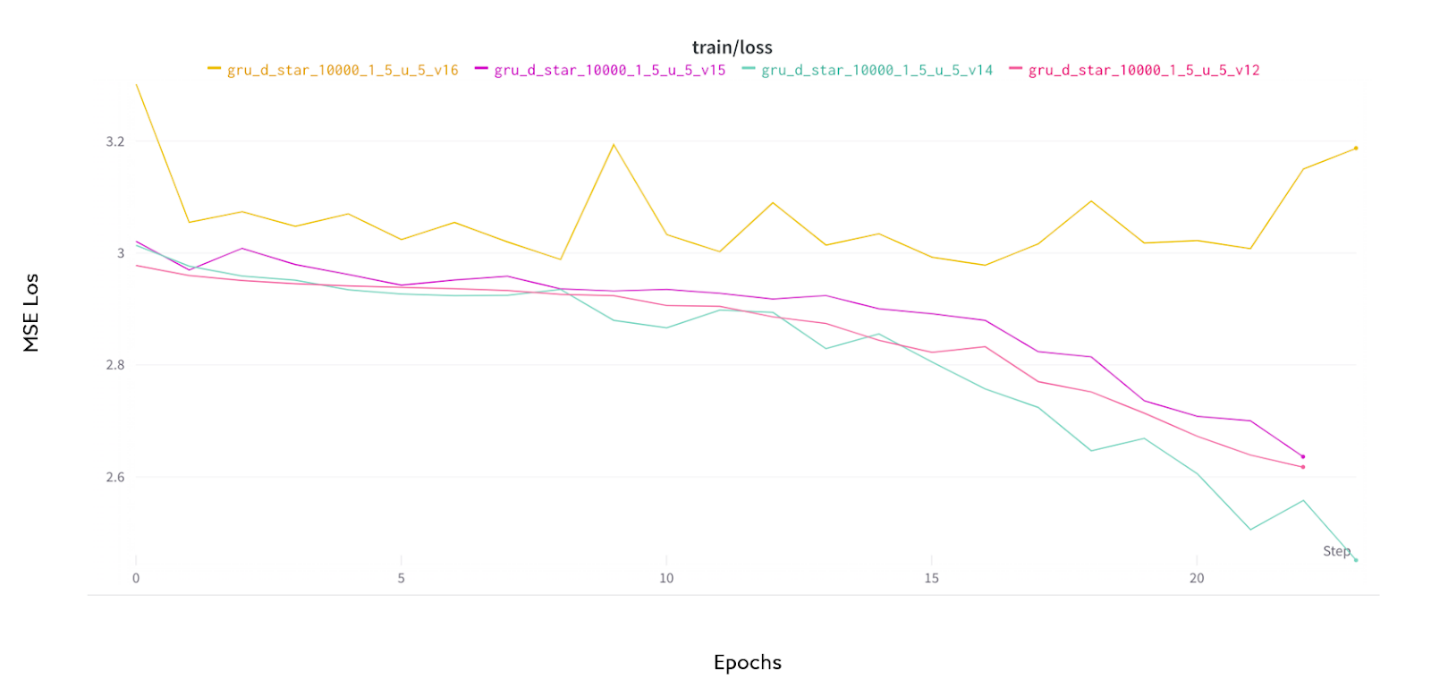
There are two simulation conditions generated as shown in the Table 6.1. star2 has large number of packet flow than star1

**6.2 NUERAL NETWORK MODEL**

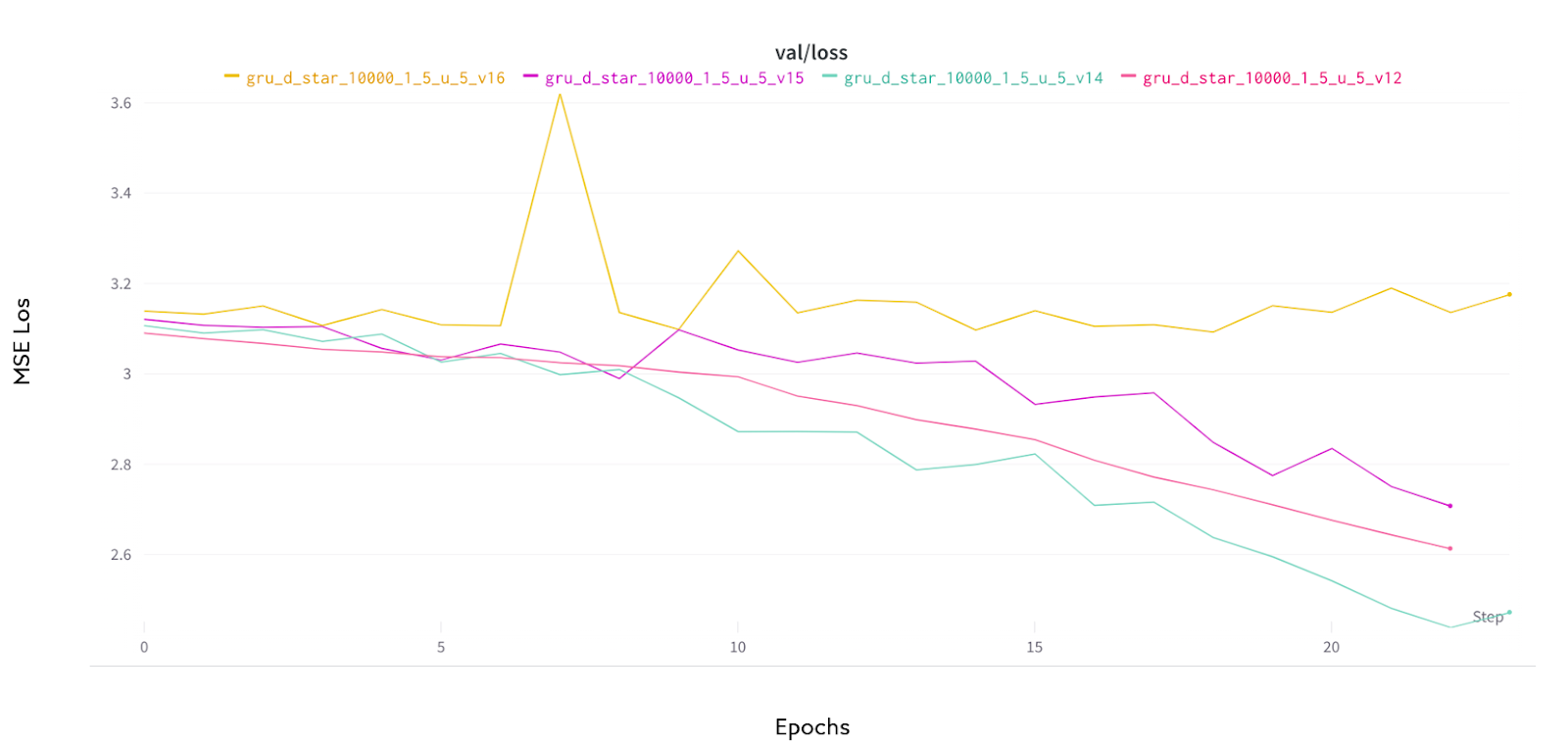
The model is trained using different hyperparameters and the better model is trained for more epochs and finalized for exporting. For the neural network two metrics are used: Mean squared error (MSE) and cosine similarity. Mean square error is the measure of square of errors. The training and validation plots are plotted below in Fig 6.2 and Fig 6.3. Initially the model is trained by varying LR, number of hidden parameters in RNN (RNNH) and number of layers in RNN (RNNL). The comparison table is provided in Table 6.2

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Colour** | **LR** | **RNNH** | **RNNL** | **train/cos\_sim** | **train/loss** | **val/cos\_sim** | **val/loss** |
| blue | 0.001 | 1024 | 1 | 422036.09 | 2.45 | 444539.53 | 2.47 |
| purple | 0.001 | 512 | 2 | 404367.68 | 2.63 | 430314.43 | 2.7 |
| pink | 0.0001 | 1024 | 1 | 422091.09 | 2.61 | 435415.65 | 2.61 |
| yellow | 0.01 | 1024 | 1 | 398403.625 | 3.18 | 443747.71 | 3.17 |

**Table. 6.2 Initial Hyperparameter Tuning**



**Fig. 6.2 Train Loss with initial tuning**



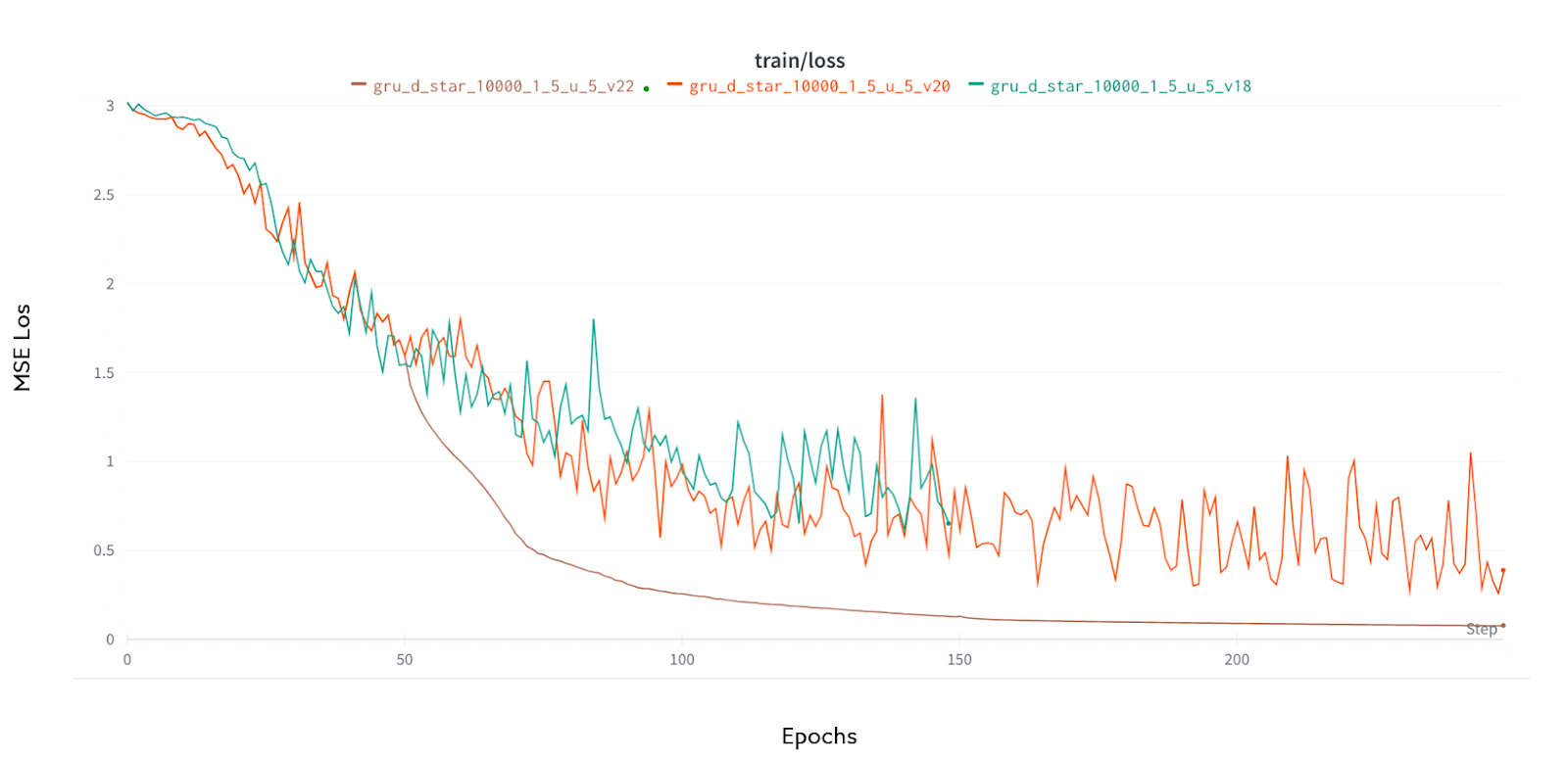
**Fig. 6.3 Validation Loss with initial tuning**

It can be seen from the above values that training with 0.01 or 0.0001 is not as much as optimal than running the model with 0.001.

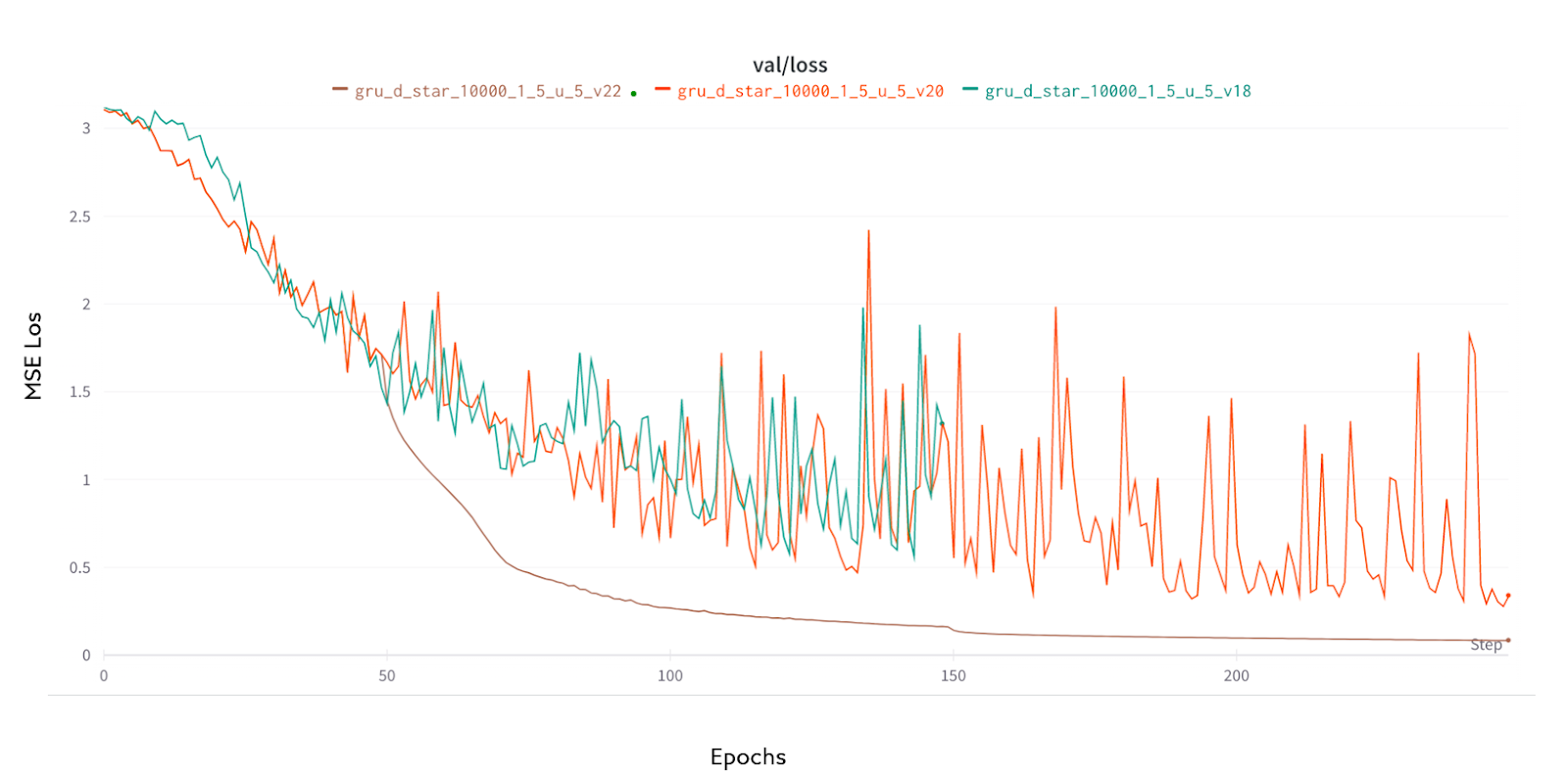
Tests were run again for selected models shown in Table 6.3 the value fluctuates a lot after a certain number of epochs the loss value fluctuates so a learning rate scheduler is used. With the LR scheduler the model achieved a validation loss of 0.08. The training and validation loss are shown in Fig 6.4 and Fig 6.5.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Colour** | **Epochs** | **LR** | **RNNH** | **RNNL** | **train/cos\_sim** | **train/loss** | **val/cos\_sim** | **val/loss** |
| Brown | 250 | 0.001 | 1024 | 1 | 4575740.5 | 0.077 | 4640719.5 | 0.085 |
| Orange | 150 | 0.001 | 1024 | 1 | 2500711 | 0.83 | 2727302 | 1.216 |
| Blue | 150 | 0.001 | 512 | 2 | 2613263 | 0.65 | 2766223 | 1.31 |

**Table. 6.3 Hyperparameter Fine Tuning**

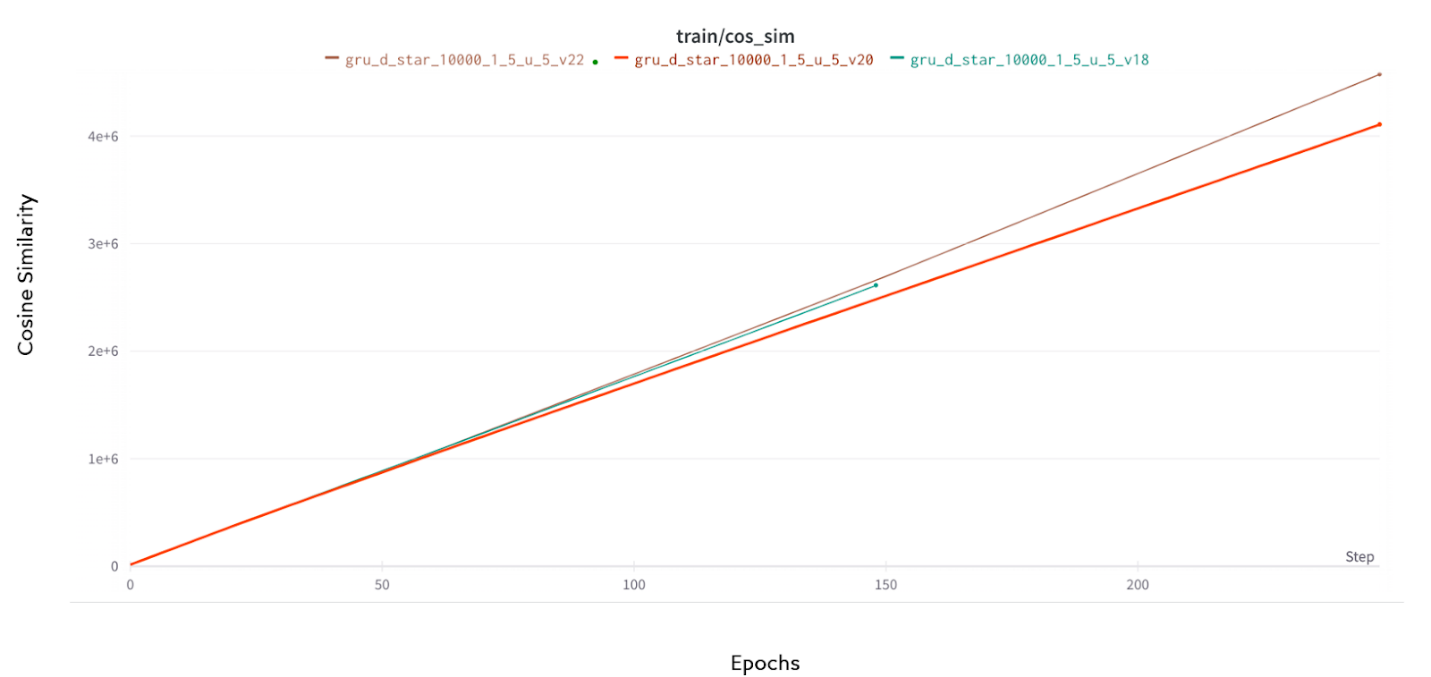


**Fig. 6.4 Train Loss Fine Tuned**

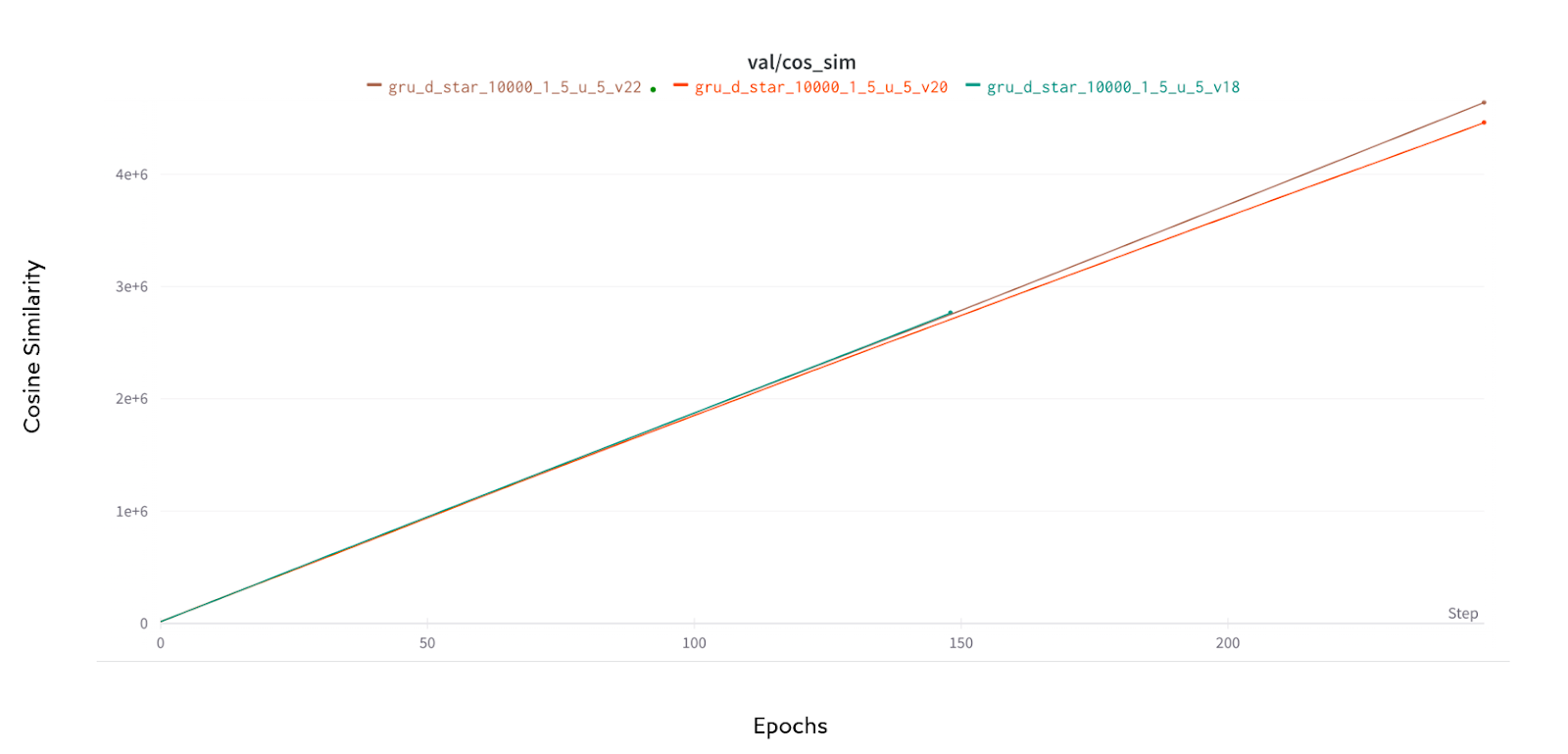


**Fig. 6.5 Validation Loss Fine Tuned**

Another parameter which was used to validate the model is cosine similarity; it can be seen in Fig 6.6 and Fig 6.7 the selected model (brown) outperforms the other model.



**Fig. 6.6 Training Cosine similarity**



**Fig. 6.7 Validation Cosine similarity**

**6.3 SHORTEST PATH IMPLEMENTATION**

The neural network model predicts the future metrics with which the edge weights are calculated and then used for the shortest path algorithm. The algorithm is evaluated against Dijkstra’s algorithm. The average delay, jitter and bandwidth is calculated for simulation run for 15 mins (900s). The metrics are calculated in the same way the data is generated. As per the Table 6.3 delay is reduced for GRU based weighted shortest path algorithm.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Predicted weights** | | | **Equal Weights** | | |
| **Name** | **Delay (micro second)** | **Jitter (micro second)** | **Throughput (bytes per second)** | **Delay (micro second)** | **Jitter (micro second)** | **Throughput (bytes per second)** |
| star1 | 588.27 | 1014.89 | 2908902.52 | 10863.17 | 8355.24 | 2019938.33 |
| star2 | 11863.76 | 12394.79 | 1967047.42 | 15683.16 | 17991.91 | 1256734.12 |

**Table. 6.4 Comparison table between predicted weight and equal weight Dijkstra’s Implementation**

**CHAPTER 7**

**TOOLS AND TECHNIQUES USED**

**7.1 GO LANGUAGE**

It is suitable for Multithreading applications [12], separate threads are used for sending packets from the host. Used in building host applications (listening to specific ports, generating Iperf and ping commands and forming UDP packets to send to controllers in separate threads).

**7.2 ONOS CONTROLLER**

It is a commercial grade scalable open flow-based controller. It is built in Java and can work with different southbound API [4] like OpenFlow, OF-Config, NETCONF, BGP, SNMP.

**7.3 MININET SIMULATOR**

Used to simulate network conditions with multiple hosts and switches. The algorithm would be tested and validated in the New York city network [3], Full Mesh and partial Mesh network.

**7.4 LANGUAGES USED**

* **Python:** It is used to build a neural network model to predict traffic nature using GPU. Pytorch module is used for training and generating the GRU Neural network model
* **Java:** Used to build ONOS application
* **Shell Script:** Used to generate traffic in mininet

**CHAPTER 8**

**CONCLUSION AND FUTURE WORK**

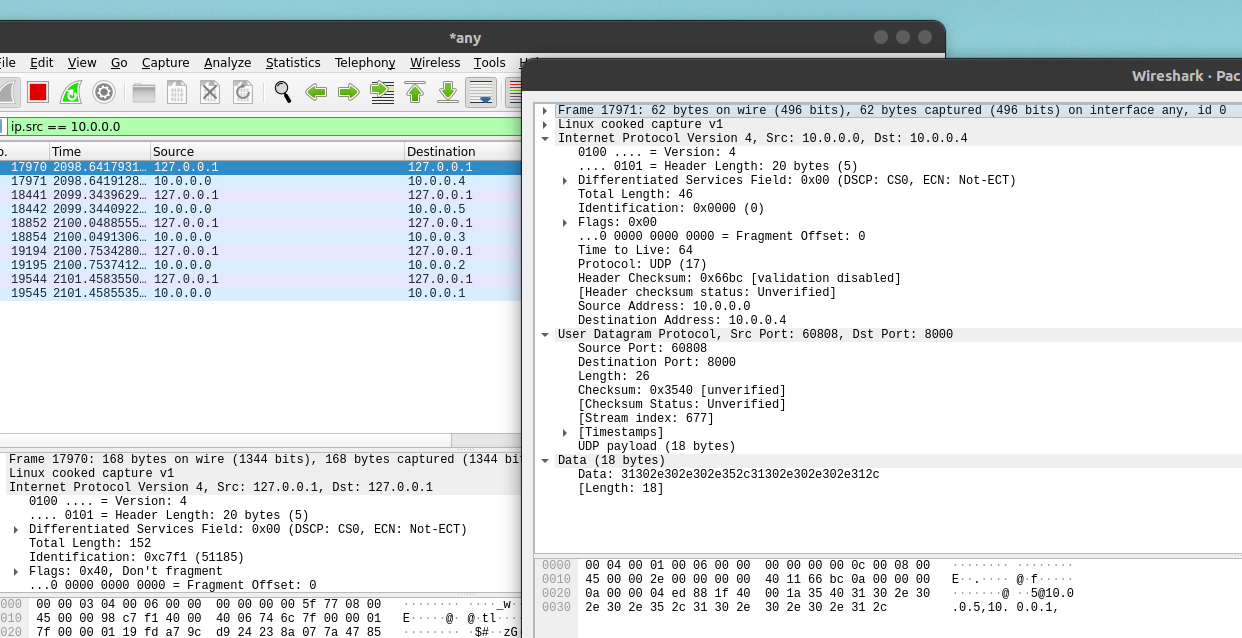
**8.1 CONCLUSION:**

This work optimizes the network traffic parameters in software defined networks by analysing and predicting future traffic thereby giving the ability to controller to realize sudden change of traffic conditions and providing an optimal route. This work used GRU neural network to predict future traffic which is normalized using entropy. This amplifies the effect of parameters which reflects traffic conditions better comparatively. The model is evaluated and compared with equal weights shortest path algorithm. It is proved that the network parameters delay and jitter is reduced in this technique and throughput of the network is increased.

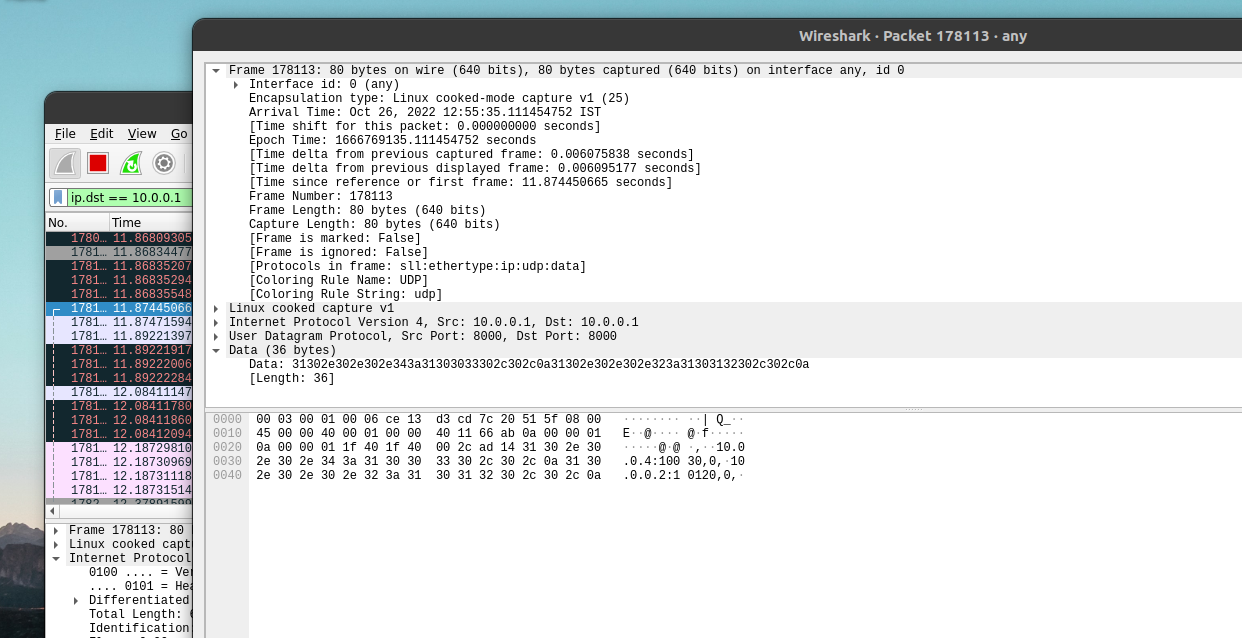
**8.2 FUTURE WORK:**

This model may react according to sudden traffic changes but would fail in case of link failures. The algorithm should include the capabilities of reducing the recovery time in case of link failures. Other parameters can be considered for predicting the traffic. Further priority of packets can also be considered to optimize traffic flows.

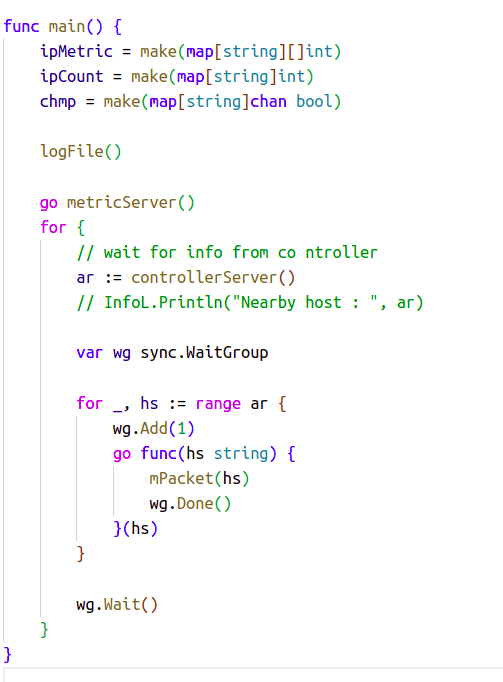
**APPENDIX**

****

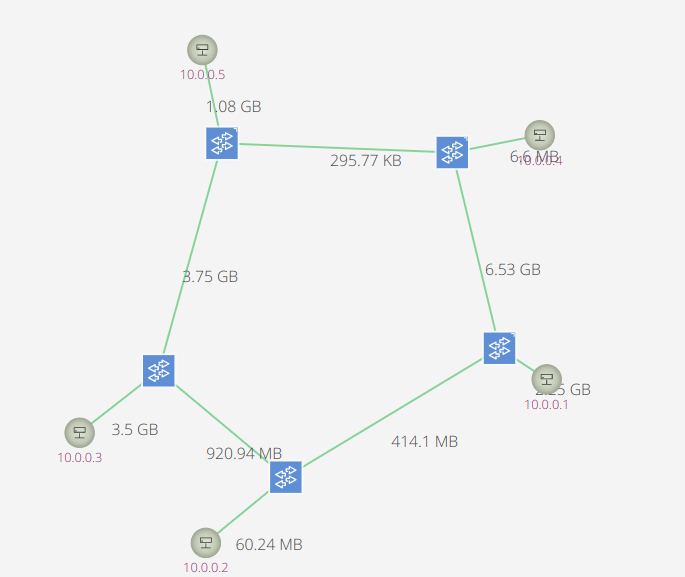
**Fig. A.1 Packets from Controller**



**Fig. A.2 Packets to Controller**

****

**Fig. A.3 Host application code screenshot**

****

**Fig. A.4 Output Flow**

**CHAPTER 9**

**REFERENCES**

[1] Shirmarz, A., Ghaffari, A. Automatic Software Defined Network (SDN) Performance Management Using TOPSIS Decision-Making Algorithm. *J Grid Computing* 19, 16 (2021). https://doi.org/10.1007/s10723-021-09557-z

[2] H. Yao, X. Yuan, P. Zhang, J. Wang, C. Jiang and M. Guizani, "Machine Learning Aided Load Balance Routing Scheme Considering Queue Utilization," in *IEEE Transactions on Vehicular Technology*, vol. 68, no. 8, pp. 7987-7999, Aug. 2019, doi: 10.1109/TVT.2019.2921792

[3] Huang, L., Ye, M., Xue, X. et al. Intelligent routing method based on Dueling DQN reinforcement learning and network traffic state prediction in SDN. Wireless Netw (2022). <https://doi.org/10.1007/s11276-022-03066-x>

[4] Mohammadi, R., Javidan, R. EFSUTE: a novel efficient and survivable traffic engineering for software defined networks. J Reliable Intell Environ 8, 247–260 (2022). <https://doi.org/10.1007/s40860-021-00139-0>

[5] BinSahaq, A., Sheltami, T., Mahmoud, A. et al. Fast and efficient algorithm for delay-sensitive QoS provisioning in SDN networks. Wireless Network (2022). <https://doi.org/10.1007/s11276-022-03028-3>

[6] Zhang, Y., Chen, M. Performance evaluation of Software-Defined Network (SDN) controllers using Dijkstra’s algorithm. Wireless Netw (2022). https://doi.org/10.1007/s11276-022-03044-3

[7] A. Santos Da Silva, C. C. Machado, R. V. Bisol, L. Z. Granville and A. Schaeffer-Filho, "Identification and Selection of Flow Features for Accurate Traffic Classification in SDN," 2015 IEEE 14th International Symposium on Network Computing and Applications, 2015, pp. 134-141, doi: 10.1109/NCA.2015.12.

[8] D. Jayasinghe, W. H. Rankothge, N. D. U. Gamage, T. C. T. Gamage, S. D. L. S. Uwanpriya and D. A. H. M. Amarasinghe, "Network Traffic Prediction for a Software Defined Network Based Virtualized Security Functions Platform," 2021 IEEE 12th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON), 2021, pp. 1083-1088, doi: 10.1109/IEMCON53756.2021.9623169.

[9] https://opennetworking.org/sdn-definition/ (2022)

[10] Dana Hasan and Mohamed Othman “Efficient Topology Discovery in Software Defined Networks: Revisited” ,2nd International Conference on Computer Science and Computational Intelligence (2017)

[11] https://obkio.com/blog/how-to-measure-jitter/

[12]https://www.analyticsinsight.net/top-10-modern-programming-languages-that-are-not-up-to-the-mark/ ( sept 12 2022)

[13] Y. -D. Lin, H. -Y. Teng, C. -R. Hsu, C. -C. Liao and Y. -C. Lai, "Fast failover and switchover for link failures and congestion in software defined networks," 2016 IEEE International Conference on Communications (ICC), 2016, pp. 1-6, doi: 10.1109/ICC.2016.7510886.

[14] .P. Chen, “Effects of normalization on the entropy-based TOPSIS method,” Expert Systems with Applications, vol. 136. Elsevier BV, pp. 33–41, Dec. 2019. doi: 10.1016/j.eswa.2019.06.035.