EAI Endorsed Transactions
on Internet of Things

Research Article **EAI.EU**

A Reliable Load Balancing Fault Tolerant Multi-SDN Controller approach in a typical Software Defined Network

Sharathkumar S[1,*] , N Sreenath[2]

[1]Research Scholar, Department of Computer Science and Engineering, Puducherry Technological University, Puducherry
[2]Professor, Department of Computer Science and Engineering, Puducherry Technological University, Puducherry

Abstract

INTRODUCTION: In the 21st Century, we have an emerging, modernized networking approach for efficient transfer of data in the form of packets i.e Software defined Networking (SDN). It is often recognized as the network paradigm acquired by an isolated independent control plane by the means of the underlying networking interface. As the communication medium is getting rapidly increased, the usage of the number of resources are also increasing in the same ratio, this results to connectivity problem in communication networks as well. If the development in technology is gaining more interest, the resource utilization will also increase.
OBJECTIVES: In the proposed work here, we need to establish the connection between the client (or the end user) and the server by calculating the network performance using the parameters such as server response time and the throughput of the network by the method of load balancing.
METHODS: This paper aims to identify the new methods or techniques to improve the performance of SDN with the increase in reliability, specifically required to improve the real-time data delivery. Alternatively, a reliable disjoint, distributed Multi-SDN controller will be used, to collect the updated network information like packet size, connectivity, packet flow etc. The methods involved are ideal controller optimization technique, distruption acquiscent technique and Maximum flow shortest route technique.
RESULTS: From the results obtained by the simulation, the Reliable Load balancing Fault Tolerant Multi-SDN Controller(RLFTMS) approach is more reliable, fault tolerant compared to the other existing traditional mechanisms.
CONCLUSION: Load balancing based on the server response time, throughput is being compared , this improves the server workload ditribution among various datacenters. Ultimately the customers or clients will have a great experience in terms of the reducing response time and increase in the throughput.

Keywords: SD-WLB, LBBSRT, Round Robin (RR), Random, Server Response Time, OpenFlow, Load Balancing, Throughput, RYU Controller, IoT.

Received on 12 January 2022, accepted on 18 January 2022, published on 02 February 2022

doi: 10.4108/eai.2-2-2022.173295

[*]Corresponding author. Email: Sharathkumar.s@pec.edu

1. Introduction

In recent years, computer networks plays an important role in the communication system and that too in Internet of Things (IoT)[11]. It is gaining more attention among the young and dynamic researchers to a large extent. The advanced features in communication are taking place in this modern era, users are gaining interest in the field of communication networks. Overall, the growth done in the field of research and development has reached its glory. This mechansim is getting popularity and lot of people are showing interest to do research under the area of Wireless Networks and in particular about SDN, since the growth in network traffic is reaching its glory and in particular future networks like 5th Generation (5G) and 6th Generation (6G) networks.

This motivates the researchers to build an efficient networking architecture named SDN which should be easily managed, programmable and having centralized accessibilty. This allows all the applications or the network operators to get the control of the whole network and make easy to understand the behaviour of the entire network as well. SDN makes the network software more useful [12].

In order to change the distributed approach of keeping control plane and data plane together, the SDN approach will make it centralized i.e making one controller acting in the control plane and all the other devices in the data plane are connected to it , so that the network programmability becomes simple and we can have separate data traffic and control traffic. In other words a centralized controller acting at the control layer will take care of the end users in the form of hosts depicted in the data plane. This helps in easy and simple programming ability of SDN [13].

Here, we focus on one of the important concept called fault tolerance. In the context of fault tolerance, SDN provides important opportunities with its central view on the whole network. Therefore, a fault tolerance mechanism is required which reduce single point of failure in the network by using multiple controllers. In SDN, considering all the three layers or planes, the fault tolerance is acting in all these layers , we need to take care about how to handle the fault in all the planes of SDN. The problem existing in **data plane** includes link failures and switch failures whereas the **control plane** takes care of the failure in connection between switch and controller or controller failures. The main function of the **application plane** is to detect the failure in an application which in turn will affect the connectivity established through northbound Application Program Interface (API) and it will spread to the other applications gradually[14].

As a significance, this mechanism also increases the scalability, reliability and high availability of services in the network.

Further, controllers are categorized into two types: **centralized** and **distributed** [1]. We need to understand the respective architectures of both types of the controllers. According to the survey done until now, the distributed controller approach where the control plane and data plane are coupled together faces the synchronization problem, connectivity issues, no proper fault management activities etc. In the centralized approach, the remote SDN controller is used to act as a centralized system and connects all the networking devices and will achieve complete access. This approach decouples the control plane and data plane, where in a single controller is acting like a master and the networking devices as slave. The principle resembles the master-slave approach. But, there is a limitation in SDN or centralized controller approach i.e they suffer from a single point of failure and also they will face the problem of resource utilization, scalability issues. In order to avoid this limitation, we make use of the multi-controllers as a backup controller and work cooperatively**.**

Further, the control plane consists of a suitable SDN controller in a typical SDN environment. The controller situated in the middle layer of this architecture consist of network operating system (NOS), helps to provide any network applications and also software-based services [15]. Further, single centralized controller, which is responsible for managing the entire network system is situated in the control plane. This can be both advantage and disadvantage for a network. One centralized network will monitor the whole network, this solves the scalability problem. Futher, there is a drawback in SDN[16], if a single point of failure occurs then the entire network fails. To overcome this, an updated mechanism should be proposed by having a Network Information Base (NIB) which is used to store all the switches acting in the data plane in order to handle the packets and also through the ingress port of the switch the packets will be arriving and they are matched to the flow accordingly[17].

Here we are making use of the Mininet tool and SDN controller named RYU[8][9] a free source and was developed by a telecommunication company in Japan named Nippon Telegraphy and Telephone(NTT). The word “RYU” means “flow” in their language. The main function of this controller is to provide easy and controlled flow of data, so that we can get a interactive networking platform. This controller supports some protocols like Netconf, Openflow and even managed properly as well. The language used and developed in RYU controller is python and it is the most efficient and useful coding language for implementation of any required work related to SDN.

Further, due to the structural characteristics of SDN, the function of the switch is simpler and it is only responsible for data forwarding, makes the switch design simple [18]. The essence of the SDN concept is to separate the control layer from the data layer, which simplifies network management, improves network flexibility and scalability. In large networks, the good placement of the controllers makes the best use of the existing structure of links between the network nodes. One centralized controller is not enough to control all the nodes in a large SDN, as its performance is limited, and synchronization between the controller and every clients or nodes in SDN will not be in real time and there exits a signal propogation delay [19]. Because, single controller works for a small network, it is not suitable for the large networks, in order to improve the scalability in the network multiple controllers can be used in the entire network. In the multiple SDN controller scenario, either there can be a replication of network state information or it may be shared among all the controllers existing in the network. We should achieve synchronization among the controllers as per the periodic time to finally get consistency in the network [20].

In spite of applying various methods like testing, decoding, verification, validation, certification the rate of large-scale failures is more in the production networks and remain less suspectible [21]. The availability of resources to a load balanced network [2] is obtained by the method of optimization using the various process by reducing the response time, gradual raise or increase of throughput and reduce in the overloading of resources in the intended network**.** Multiple controllers are reserved for backup but we should take care about the scalability and availability issues in this mechanism. If there is any failure in the controller, the failed controller is replaced with an alternative controller which is already there in the queue. In order to achieve more reliability, we require many instances of the controllers and also have a well stock of alternative controllers in the form of backup and have the clear information about where to install the alternative controllers available [22]. Distributed controller handles the whole network, while maintaining complex tasks such as quality of service, security and load balance [23]. Here, in order to fullfil the available demands of the next generation networks, a logically centralized network is used, this even gives the backup for the network administrators to enable and regulate the network-wide traffic flows. Through this centralized approach we can create the dynamic network topologies defined as in the data centers and also policy based routing can be implemented in service providers or enterprise networks [24].

As shown in [3] [4] the motivation should be to implement the architecture of SDN using a RYU SDN controller which is a free source, mainly used for the analysis of the network and to evaluate in deep, performance analysis of SDN architecture for various parameters like the number of packets transmitted, packets received, throughput, bandwidth, round trip time etc.

In this paper, for a SDN network a proposed method for load balancing is explained in a new way by optimally load balancing the network by reducing the duration of time in response of the server. We define the metric for load balancer i.e the **server response time**, it is the time or duration of the requests commenced that means end users accepts the request and gets the response in the form of reply from server. If the load is heavy or high, then the server response time will take long time. Our approach should be efficiently help to calculate the server response time by using the load balancing method.

The main work carried out by this paper includes,

- Using the SDN architecture, the performance metric like real time server response time is calculated w.r.t specified controller to design an efficient, load balancing scheme in the network.
- Realize the network traffic for the potential implementation of load balancing scheme using the suitable SDN Controller.
- Compare the performance metrics like the server response time , throughput and provide the effectiveness of the proposed scheme over the existing traditional approaches.

Finally, a reliable, disjoint and fault tolerant multi SDN controller has to be developed.

2. Related Works

In this section, we review the other researches related to the work we propose.

In [1], the work mainly focuses on Data center Networking and uses floodlight controller, uses load balancing algorithmic approach, but mainly it fails to use other controllers and no response time is considered with no dynamic methodology adopted to calculate the throughput of the system.

As in [2], SDN flexibility is used for load balancing describing real-time measurement of server response and it is not considering the energy saving issues and the balanced load is not achieved. In [3][5], the comparison of round-robin, random etc. Startegies are considered but no server load and no real hardware is used and the controller is limited to POX alone.

In [4], Intercloud manager (ICM) in SDN is used to allocate network flows in cloud for monitoring and decision making in cloud network. This method fails to improve the performance of the SDN network and also the energy-awareness is restricted to small scale.

A dynamic server load balancing technique is proposed using RYU controller in [6], with Mininet and Raspberry pi switch compared with Random, Round-robin load balancing strategies. But the testing of code done only for reliability and resource utilization, throughput and server response time is not considered here.

The performance test analysis on SDN controller is done in [7]. Parameters considered are throughput, Round Trip Time (RTT) using Mininet, RYU controller, and performance on bandwidth, throughput, RTT, jitter, packet loss using RYU are calculated. But the restriction here is only adopted to default topology and other topologies like linear, single, tree topologies are not taken into consideration.

RYU controller alone is used considering the bandwidth, throughput and RTT for custom topology between any 2 nodes in [8]. Comparison with other controllers the work carried out here is by considering the other networking parameters as well.

Deployment of Low-cost high performance open testbeds are carried out in [9] considering controllers like ONOS, floodlight, RYU etc. with low-cost open switch. But the networking functions like Quality of service (QoS), dynamic flow installation, server response time, throughput is not considered.

The SDN architecture is centralized, focus is on traffic engineering load balancing in data plane. In [10],bayesian network scheme is used to choose alternate path but no multi-controller scenario is used considering the standard network topologies to reduce the SDN controller overhead.

Sminesh et al [25], in their work partitioned the network using a modified-Affinity Propogation (AP) clustering algorithm and provides the input by calculating the distance between switch and other components, as well as uses link bandwidth between each .There is no mechanism of load balancing w.r.t server or the response time.No involvement of any external controllers like Mininet is involved, so the network traffic, performance could be analyzed. No efficient solution for placement of controllers and optimality is achieved w.r.t to the placement. Only the distance and bandwidth are considered as the input and no other parameters are considered. The major work involved here is about clustering and there is no other methods used for the load balancing of SDN controllers. Failure of internal controllers is not been involved and no solution for the fault recovery and management.

Alenazi et al [26], developed a new nodal metric, nodal disjoint path (NDP), which measures a node's importance in terms of its diverse connectivity to other nodes. NDP determines the locations of the k- controllers to increase network robustness against targeted attacks by using US-based fiber-level networks and evaluate centrality-based attacks and random failures. The NDP-global algorithm evaluated here provides better network resilience in the face of centrality-based attacks and random failures. The results also indicate that the NDP-cluster algorithm has a delay performance comparable to that of the k-median algorithm and provides higher network resilience. There is no work carried out w.r.t the placement of software-defined SDNs.

Ruaro et al [27] proposed a multi-objective management based on a distributed SDN (D-SDN) architecture (SELF-SDN). Fault-tolerance experiments highlight the simplicity of the SDN paradigm to recover from faults in the NoC, not requiring additional hardware. Results related to multi-objective management demonstrate the fast reaction time of SELF-SDN to recover the communication latency faced to QoS loss and faults. But the work is not satisfied and no work is carried out in respect to extending management objectives by addressing power and energy by turning off unused CS routers and security.

Phemius et al [28], proposed DISCO, an extensible DIstributed SDN COntrol plane able to synchronize with the distributed and heterogeneous nature of modern overlay networks. A DISCO controller manages its own network domain and communicates with other controllers to provide end-to-end network services by using AMQP protocol and evaluates inter-domain topology disruption cases. But more techniques helpful to extension for additional resilient and recovery mechanisms can be done so that a controller can take the control of switches from a neighbor domain on the fly in case of failure.

Yang et al [29], developed the SDN controller placement problem for single-link and multilink failures, respectively. For single-link failures, we develop a heuristic algorithm to address the controller placement problem. For multi-link failures, an efficient method Monte Carlo Simulation is being used to reduce the computational overhead. We conduct experiments with real network topologies, and the simulation results show that the heuristic algorithm can save significantly more time than the optimal algorithm, while achieving good performance and ultimately improves the time complexity of the algorithm. However, with a larger scale failure analysis, and a need for more precise prediction mechanism for those vulnerable links, nodes, and controllers are required.

Satheesh et al [30], developed a priority-based model using SDN to control the flow of data packets over the network, gives assurance to the bandwidth enforcement, and reallocation is made through virtual circuits. The network behavior of the system is continuously monitored through the machine learning model for normal and abnormal traffic data transmission to detect anomaly intruders. Flow-based machine learning (FML) model with SDN act as an intelligent system to limits the throughput virtually through the flow of reserved bandwidth and make use of extra bandwidth, which presents more than the utilization bandwidth for priority-based applications with minimal cost while compared with the traditional methods. However, steps have to be taken to improve network traffic in a real SDN environment.

In [25], a solution for controller load balancing, considering the dynamic load of each controller by developing an external SDN controllers has to be defined. As of [26] the proposed algorithms are to be applied to the placement of software-defined SDNs. In the paper [27] the concept where the requirement on extending management objectives by addressing power and energy by turning off unused CS routers and security is discussed. In [28], the technique has to be extended for additional resilient and recovery mechanisms so that a controller can take the control of switches from a neighbor domain on the fly in case of failure, in [29] a larger scale failure analysis, and provide a more precise prediction mechanism for those vulnerable links, nodes, and controllers are required and as of [30] some mechanism w.r.t steps has to be taken to improve network traffic in a real SDN environment.

Hence to overcome the above mentioned issues, a fault tolerant, multi SDN controller has to be developed.

3. System Architecture

RYU controller provides software components by making use of well-defined API's, this makes developers or the network admin to create new network management and control applications. RYU supports the well defined protocol named OpenFlow and with versions of 1.0,1.2,1.3,1.4 and 1.5 and the controller is implemented in the python language.

RYU SDN Controller consists of three layers or 3 planes. The lowest layer is the infrastructure or physical layer which consists of various physical and virtual devices interconnected via internet for the purpose of communication. It also consists of different devices placed within the same plane.

The middle layer or the 2nd Layer known as control layer or the network lyer consists of network devices and hosts ,both connected via Northbound API's and Southbound API's. Further, this layer is used for flow control mechanism i.e data traffic from one device to another device to provide the stability in the network without any network traffic overhead.

The interface between physical and control layer is done with the guidance of Application programming interfaces (API) i.e southbound API's like OpenFlow, OF-config etc.

The topmost layer is the application layer usually consists of network logic applications and business applications. The interface acting between the application layer and network layer is issued with the consultation of API's known as Northbound interfaces like OFREST, Firewall, Quantum etc.

The RYU Controller used here, make use of OpenFlow protocol to interact with the forwarding plane consisting of switches and routers for handling the traffic flows. The testing process is carried out using Openvswitch and also supported by some companies such as Centec, HP, IBM etc.

All these layers have the similar goal or target to collect the network intelligence at one fixed area or the place named the centralized controller. The objectives gathered here by the controller runs the algorithms and orchestras the new rules used by the controller.

3.1 General System Architecture

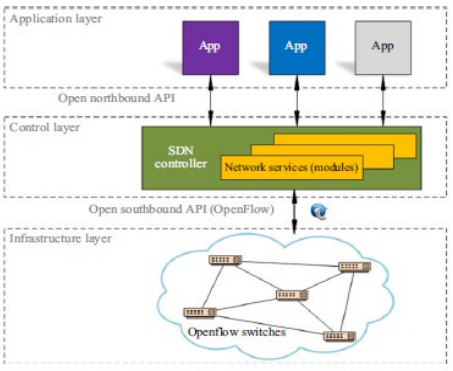


Figure 3.1: Layered Architecture of SDN

The Figure 3.1 depicts a layered Architecture of the SDN. This architecture comprises of a standard 3-tier namely the bottom infrastructure layer, a middle Control layer and the above Application layer.

Various topologies and the devices are constituted by the lowest layer which is formed of OpenFlow switches and connecting devices in any network. It is also known as forwarding plane or data plane and it's main responsibility is data packet handling initiated by the client or the user via any linked devices to obtain the network traffic and get forwarding information based on instructions from the control plane.

The control plane's main responsibility is decision making that is handling of packets and forwarding of the data at the linked devices properly across the network. The SDN controller is present or included in the middle control layer and it is used for the overall operation management of the entire system. The controller used here is **RYU.** All kind of network services related to the packet flow, data transmission, synchronization and updating of forwarding tables and even packet handling policies also reside in this layer.

To establish the connection and the transfer of data in between control layer and infrastructure layer is done via Southbound API and uses a well known protocol called OpenFlow protocol, which is used to maintain flow rules for forwarding of packets from controller. There is a fair independence between the controllers used to overcome any network issues or transmission issues and also resilience of the network in the control layer.

Finally, the application layer contains the information of any network services used and accessed in the entire network via the useful network operator. Additionally, this plane consists of the network behavioural information in terms of its applications and services. Some of the routing processes acting in the control layer where no applications are considered directly or even support this kind of operation are not considered to be the part of this layer. The connection or communication between the control layer and Application layer is achieved by Open Northbound API and the RESTful API.

3.2 Proposed Working Model

The working model of a typical RYU SDN framework architecture is depicted as shown in the figure 3.2.

The first (Topmost) layer is the Application layer. In this layer the testing and logical design of the application programs is thoroughly checked and made implemented. All the applications are acting in this layer including the End user applications, the Operator used and the OpenStack cloud orchestration. This layer is connected with the Network layer or the control layer through RESTful API, Northbound API, RestAPI and User defined API. Programming system arrangements, administrations and even the switch packet processing, brings numerous new potential outcomes to the system administration

The second middle layer or the intermediate translation layer is the Network layer where the Framework of RYU SDN controller is defined. Inter-controller communication exclusively refers control layer own operations, rest of the operations are affected by the other layers.It consists of Built-in RYU applications, Firewalls, Libraries including the Toplogies used. OpenFlow protocol with versions like 1.0,1.2,1.3 etc. and Non OpenFlow protocols Netconf, VRRP etc.

In the network layer both, Ideal Controller Optimization approach and distruption acquiscent techniques are included.

The lowest layer or the bottom most layer is the Infrastructure layer. It mainly includes the OpenFlow switches and Network Devices. Fault tolerance issues like link failure, node failure normally available in traditional network are part of this layer. But in SDN due to centralized management and programmability leads to new challenges. The failure is detected at link or node level. SDN permits novel kind of fault recovery arrangement. In SDN numerous arrangements broaden the southbound protocol conduct as well as repurpose header fields to help fault recovery systems.

The techniques or approaches are synchronized with standard SDN Model as shown in the Figure 3.2.

The 3 layered proposed methods of RLFTMS is as shown in the Figure 3.2. The topmost approach is the novel ideal controller optimization approach where the sub controllers used in the approach should be optimal and also it should take care of the controller load by checking the maximum capacity it can handle for any process to happen so that no problem should happen in the connectivity and also check the ping connectivity so that to verify whether all the connections are up to date and end to end connection can be done smoothly.

The middle layered approach in the Figure 3.2 is the novel disruption acquiescent technique. This technique is majorly used for checking the connectivity between the sub controllers and switches and in turn connected to the hosts. Further, to check the behaviour of the network i.e. normal or not especially the nodes in IoT networks. This approach uses Micro-cluster Outlier detection method which is based on the conditional entropy approach that is used for incorporating the network traffic flows, separation of the data traffic and control traffic using a deep packet inspection where the network behaviour can be easily judged and also we can check any security lapse has happened like the DDOS attack has happened or not. It helps us to eliminate the vulnerability of attacks as well. Ultimately, the unnecessary link failures in the network can be eliminated. This solves both the single link and multi-link failures by using distribution decision scheme containing load information and using the nearest switch which has lowest active nodes. In the multi-link failures, the controller cannot do it single-handed, so it will allow the nodes to migrate to the neighbouring switch linked with the respective sub controller.

Finally, the lower layered approach which is the Maximum flow shortest technique is used to determine the secured and fastest route of the network and also to overcome the overlapping of data in the network path. This technique is developed since it uses shortest and minimal path consisting least number of links, so that any interference of the packet happens can be avoided. It will also utilize the minimum available bandwidth capacity and flow weightage through the nodes and hence eliminate any weak nodes available in the network to achieve maximum flow capacity with minimal links and by using the shortest path without any overlapping by employing suitable transfer protocol for the proposed architecture. Additionally, flexible and common switch is attached with two controllers, is in case any controller failure happens, then the concerned switch will act as the backup link and path can be re-established without any problem.

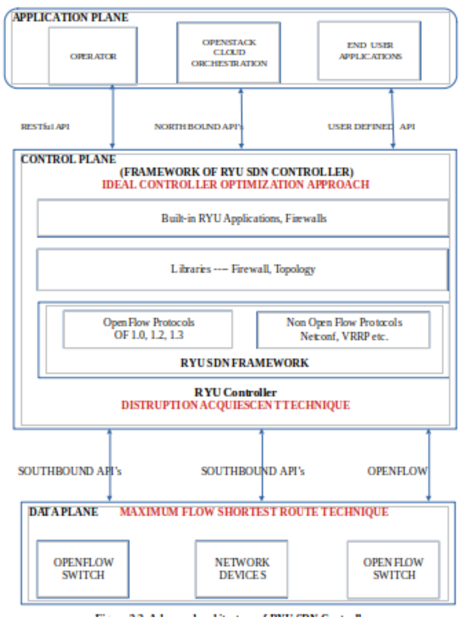

APPLICATION PLANE
OPERATOR
OPENSTACK CLOUD ORCHESTRATION
END USER APPLICATIONS
RESTful API
NORTH BOUND API's
USER DEFINED API
CONTROL PLANE
(FRAMEWORK OF RYU SDN CONTROLLER)
IDEAL CONTROLLER OPTIMIZATION APPROACH
Built-in RYU Applications, Firewalls
Libraries ---- Firewall, Topology
Open Flow Protocols OF 1.0, 1.2, 1.3
Non Open Flow Protocols Netconf, VRRP etc.
RYU SDN FRAMEWORK
RYU Controller
DISTRUPTION ACQUIESCENT TECHNIQUE
SOUTHBOUND API's
SOUTHBOUND API's
OPENFLOW
DATA PLANE
MAXIMUM FLOW SHORTEST ROUTE TECHNIQUE
OPENFLOW SWITCH
NETWORK DEVICES
OPEN FLOW SWITCH

Figure 3.2: Advanced architecture of RYU SDN Controller

3.3 List of Available SDN Controllers

There are many controllers acting in the control layer of SDN. Among them POX, RYU, OpenDaylight, Floodlight are important. The list of other controllers shown in the Table 3.1 exhibiting the comparison of various controllers with controller name, programming language, Developer, license provider and the platform support.

3.3.1 Comparison of Various SDN Controllers

The following table 3.1 signifies the comparison of the variuos SDN controllers with their programming languages used, Developer of the controller, License provider and platform support.

Table 3.1: List of available SDN controllers

Controller Name	Programming Language	Developer	License provider	Platform support
NOX	C++	Nicira Network	GPL	Linux
POX	Python	Nicira Network	Apache	Linux, MAC OS, Windows
Becon	Java	Stanford University	GPLv2	Linux, MAC OS, Windows
Mastero	Java	Rice University	LGPL	Linux, MAC OS, Windows
Floodlight	Java	Big Switch Network	Apache	Linux, MAC OS, Windows
Trema	C, Ruby	NEC	GPLv2	Only Linux
OpenDay light	Java	Cisco and OpenDaylight	-	Linux
RYU	Python	NTT	Apache	Linux

4. Proposed Work

SDN is a new concept in the modern network scenario which provides the emerging answer to provide enough flexibility that is not been achieved by the existing traditional networking systems. Several recent research studies carried out has not provided any mechanism to achieve high-level performance or fault-tolerance at high scale in SDN based networks in which high latency are the most essential issues to be taken care of, since it happens by the variation of links heterogeneity through the backup path. During no failure in the network or if it is a ideal network the backup controllers remains passive and will not be used for the process.Inorder to solve this, Load balancing during the failure becomes an important concern. To overcome scalability issue, previous research has used multi controllers but the optimal numbers of controllers to be used is not yet determined which leads to overloading and high complexity in load balancing. Many attacks lead to failure of links, by detecting those attacks, the link failure can be eliminated at initial stage itself which is not yet incorporated with the SDN plane. Generally, many techniques are employed for overcoming link failure but the complex nature of the link prediction path and too many paths and backup leads to high energy consumption and high computational time. The efficient route of the network must be determined and overlapping of data in the path must be overcome. Hence to overcome all the issues mentioned a novel technique has to be implemented.

SDN technology can help a network prepare for a successful and stable IoT deployment which can deliver the agility and flexibility that the Internet of Things demands. The architecture of the SDN is framed in a Multi Controller Framework in which a main controller is linked with optimal number of sub-controllers.

As shown in the Figure 3.2, the optimal number of sub-controllers is required to be determined which is carried by adopting a novel Ideal Controller Optimization approach which takes into account the maximum capacity of the controller, data packet size and average request of data to the controller and the required controllers are connected in series in the control plane of the network. Each sub-controller is connected to various switches which are linked with nodes. In order to identify abnormal behaviour of nodes in IoT networks a novel Disruption acquiescent technique has been introduced which uses Micro-Cluster Outlier Detection with conditional entropy approach incorporating traffic separation techniques using deep packet inspection which not only detects the abnormal behaviour but it also decides whether the abnormality has been caused by a DDoS attack and eliminates the vulnerability of attacks. Thus unwanted link failures are highly eliminated. Even, when a link failure occurs the node automatically gets migrated to the neighbouring node using distributed decision scheme with load informing strategy which considers the nearest switch with lowest active nodes. In case of multi-link failure, the controller migrates from all the nodes to another switch which is linked to other controller. Moreover, in SDN technology, delays will be the most essential issues to be taken care of, since it happens by the migration of links inaccurately which causes overlapping via the saved path. In order to do route discovery in the network and to overcome the overlapping of data in the path a novel Maximum Flow Shortest Route Technique is developed in which shortest path consisting minimum number of links is achieved thereby avoiding the interference of the packet by utilizing the minimum residual bandwidth capacity and the weightage of flow through the nodes and thereby eliminating the weak nodes to get the maximum flow with shortest path without overlapping by employing transfer protocol for the architecture.

Also, a common switch is linked with two controllers; in case of controller failure the switch acts as a backup path. In case of overloading in the main controller, a dormant controller is allocated for the routing. This results to get an appropriate flow entries that can be generated priorly to avoid the overloading of control channel and determine the latency or delay in path formation. If any link or controller failure occurs, the active controllers take over which in turn, controls the flow of data over various links and promptly replace the routing. Hence a reliable fault tolerant with no link faults or failure in an interactive, secured SDN has to be implemented.

5. The Design and Implementation of RLFTMS

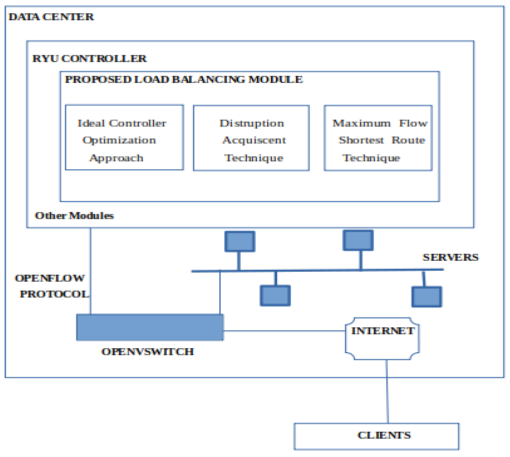

DATA CENTER
RYU CONTROLLER
PROPOSED LOAD BALANCING MODULE
Ideal Controller Optimization Approach
Distruption Acquiscent Technique
Maximum Flow Shortest Route Technique
Other Modules
SERVERS
OPENFLOW PROTOCOL
OPENVSWITCH
INTERNET
CLIENTS

Figure 5.1: Proposed Architecture of RLFTMS

The RLFTMS architecture is depicted as shown in the figure 5.1. The implemenation of RLFTMS approach is mainly carried out by the three methods proposed in the load balancing module namely Ideal Controller Optimization approach, Distruption acquiscent technique and Maximum flow shortest route technique. All the techniques are mainly used for developing the reliable fault tolerant Multi-SDN controller. The details of all the techniques is explained in detail in the section 4.

Algorithm 5.1: Ideal Controller Optimization Approach

Output: Multicontroller Framework: Linking Main Controller with Optimal Sub Controllers.

```
1.  while Main controller startup do
2.    if main controller connect to sub-controllers do
3.        for each subcontroller do
4.          Get Maximum capacity of controller
              (Max_Controller_Capacity)
5.            Get Data packet size ($DP_{size}$)
6.              Get Avg_request of data (AREQD)
7.        Control plane = Max_Controller_capacity +
                          $DP_{size}$ + AREQD.
8.        end for
9.        Connect Main Controller to Optimal sub
          controllers
10.     Sub-controllers are linked with nodes
11.   end if
12. end while
```

Algorithm 5.2: Distruption Acquiescent Technique

Output: Identification of Abnormal node(Host) behaviour in IoT Networks

```
1.  while system startup do
2.    for each node do
3.      if node behaviour = = Abnormal do
4.              Use Micro-cluster Outlier detection and Conditional entropy approach
5.  Calculate network traffic separation techniques
6.  if (Deep packet inspection)
7.        Get Node Abnormality Information.
8.        Abnormality caused from DDOS attach
9.              Eliminate vulnerability of attacks
10.               Eliminate unwanted link failures
11.           end if
12.       end if
13.     if (link failure occurred)
14.   Automatic node migration to Neighboring node
15.   Use distributed decision scheme with load
        information strategy.
16. Get nearest switch information with lowest active nodes.
17.   end if
18.    if (mult-link failure)
19.       Node Migration to other switch using Controller (Node to Switch migration).
20.       link to other sub-controllers
21.   end if
22. Store the node abnormality information and correct it.
23.   end for
24. end while.
```

Algorithm 5.3: Maximum Flow Shortest Route Technique.

Output: Determine Overlapped free route for IoT Network

```
1.  while system startup do
2.    for each node do
3.      Get backup path in each network.
4.        Calculate the path having minimum links in Network.
5.              if shortest path achieved.
6.                Packet interference is avoided.
7.                Calculate Packet interference = Minimum residual Bandwidth capacity+ Node flow weightage.
8.                      Eliminate weak nodes.
9.                  Use FTP protocol to get Maximum flow with shortest path without overlapping.
10.         end if
11.             if (Controller failure)
12.             Use Swtich as Backup path
13.           end if
14.           if ( Main controller overload)
15.                 Use dormant controller for Routing
16.           end if
17.             Assign Overlap_free_path = Control channel Overload + Path information latency
```

```
18.         Obtain Result = Overlap_free_path
19.     end for
20.   Store Result containing error_free controllers and optimal links and develop secured SDN.
21. end while
```

The ideal controller optimization approach is depicted in Algorithm 5.1. Initially, the main controller is started. Then, the subcontrollers are connected to the main controller. For each subcontroller, from line 3 to line 8 as depicted, collect and record the maximum capacity of controller (**Max_controller_capacity**), Data Packet size (**DP$_{size}$**) and Average request of data (**AREQD**). Connect all these parameters in series to the control plane of SDN. In this way, the main controller is connected to optimal subcontrollers and in turn the subcontrollers are linked with the end hosts or nodes in the network.

The second component or technique is **Distruption Acquiescent Technique** which is depicted in Algorithm 5.2, we are checking for the abnormal node behaviour in IoT. Initially, after the system is started, the node behaviour is verified. For each host or node using the approaches like Micro-cluster Outlier detection and conditional entropy approach, calculation of the traffic separation is done based on the data traffic and the link traffic. Using the method, **Deep Packet Inspection(DPI)**, we obtain the abnormality information in node. Further, the abnormality can happen from DDOS attack. Using DPI technique the vulnerability of attacks are eliminated and even unwanted link failures are also eliminated.

There exists two types of link failures, single link and multi-link failures. If single link failure occurred, then automatic node migration can happen to the neighbouring node. To do this we use distributed decision scheme with load information strategy, obtain the nearest switch information with lowest active nodes. In case of multilink failures, node migration is done with the help of controller to other switch or **(Nodes to Switch Migration).** These information are linked to the other subcontrollers. Finally, store the node abnormality and correct it to eliminate the un-necessary information.

In the third technique the **maximum flow shortest route technique** is explained in Algorithm 5.3. Here we need to determine the overlapped free route for IoT Network. Once the system is started, in each host, get the backup path in each network. Calculate the path having minimum links in network. If the shortest path is achieved, the packet interference is avoided. Calculate the packet interference by considering miniumum residual bandwidth capacity, node flow weightage. Further eliminate the weak nodes.

Further, by using **File Tranfer Protocol (FTP)** to get Maximum flow with shortest path without overlapping. If any failure of the controller, switch will having the backup path. Suppose if there is an overload of main controller, use dormant controller for routing. Additionally calculate the overlap free path by considering the control channel overhead and path information latency and store the result containing the error free controllers and optimal links and develop secured SDN.

6. System Application Model

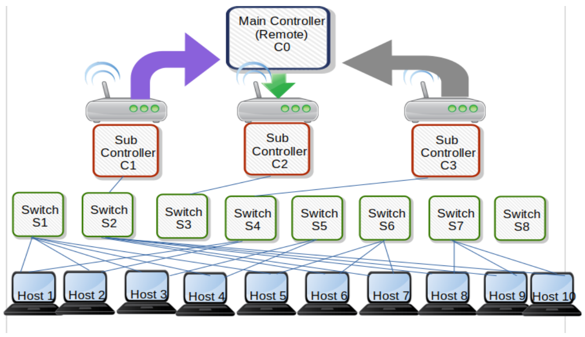


Figure 6.1: Schematic Diagram indicating the connectivity between controllers, switches and Hosts

According to the figure 6.1 application model, the main controller C0 i.e the remote controller which is RYU here, is connected to three sub controllers, C1, C2 and C3. In turn, the subcontrollers are connected to switches which are internally connected to each other. We can find the connection established between the two ends i.e the host machines with that of the switches. The hosts or clients are connected to the switches. Through the switches, hosts can connect to the controllers. Here, SDN follows the centralized approach, we have a main controller**.** If the hosts cannot get information about the data packets from the switches, then the switches will connect to the controllers and through it, there is a connectivity established between the remote main controller with the sub-ordinate controllers which are subcontrollers in this case. Now the controller will pass the information to the sub-controllers, in turn the subcontrollers will pass the data packets to switches and finally it will reach the hosts for processing of data. Like this we have a efficient and guaranteed data communication between the hosts or clients and the controllers.

The proposed system model appears to be a layered architecture consisting of the main controller(C0) in the above layer, sub-controllers below it, and number of switches connected with one more host connecting to the switches in the middle and the bottom layers. This will make the connection establishement easier and also if any error occur in the network can be easily depicted and also can be resolved as per the requirement.

6.1 Existing Algorithm to Calculate Server Response Time

Algorithm 6.1: Measuring server's response time [2]

```
1. While system startup do
2.     If current time % t == 0 do
3.         Send Packet_out to switches and record sending time $T_{send}$ ;
4.     End if
5.     If receive a Packet_in message then
6.         Parse message;
7.             If the source address of the received packet is the server, the destination
               address is the controller then
8.                 Record the time $T_{arrive}$ of received message;
9.                 Calculate the response time by the formula $T_{response} = T_{arrive} - T_{send}$ ;
10.                Store response time;
11.            Else
12.                Send to other modules;
13.            End if
14.    End if
15. End while
```

6.2 Advanced Algorithm for RLFTMS Method

Algorithm 6.2: Improved method for measuring server response time and calculation of throughput based on minimum Bandwidth

```
1. While system startup do
2.     If Main controller is remote and start_controller = = RYU do
3.         Connect and start the sub-controllers C1,C2 and C3 to main controller
4.         Check the data packet_size
5.         If data_packet_size = = 10 OR 50 OR 100
6.             Calculate the average_load of controller, No. of switches used, No. of
               connection to switches and Avg_ratio of no. of switches to no. of connection to switch.
7.         end if
8.         Print the number of controllers used
9.         While startup controllers, sub-controllers and switches
10.            Connect or link clients with respective switches
11.            Inter Connect switches with other switches
12.            Calculate the bandwidth of all clients connected to switches respectively.
13.            Check the ping reachability and make any one switch down
14.            Check the ping reachability once again after switch is down
15.            Now packet loss occurs – No ping connection
16.            Inspect the switches_B.W and obtain the minimum_B.W
17.            Add_links between the switches
18.            Create the client,switches and the link from client to switches
19.            Add the remote controller again
20.        End While
21.            Repeat Steps from 8 to 21 again in a loop
22.        Print("Creation of switches")
23.        Print("Connecting client with switches")
24.        Check the controller and sub-controller connectivity with server.
25.    Repeat Steps 8 to 21 do
26.    Check Ping reachability for calculating the packet loss
27.    Get the reachability information of all switches and obtain the new bandwidth
28.    Obtain the minimum bandwidth
29.    Finally check the ping connectivity once again
30.    Print("No packet_loss and 100% packer transmitted")
31. End if
32. End While
```

The ultimate aim of the preferred algorithm is to attach the controllers with the main-controller and the switches and ultimately to the end hosts. Check the ping connectivity and verify the bandwidth of the link and obtain the minimum bandwidth to get the least server response time and the throughput.

6.3 Simulation Study

6.3.1 Server Response Time Measurement in Real-Time

As in [2] , to measure the response time of a server there is a strategy of obtaining the it and it is as follows:

Step1: Generate Packet_Out message and pass to Switches. The controller commences once the system starts, further the controller checks the time interval say 't' to send the message i.e the transmission time and makes the initiation of sending the authenticated message and makes note of the initiation i.e the start time and the transmitted time. There should be a synchronization established between the servers available in the pool of resources and the message transmitted. The content of this Packet_out message includes the information of data in terms of packet, their source address, IP address of the controller used and also the designated receipent address attached with the respective server IP**.**

Step2: Handling of Packet_out message by Switches. Next, the intended message initiated by the controller is received by the respective OpenFlow switch, the respective switch will check the data packets by parsing and transfers these packets to the respective servers.

Step3: Reply message generated from server, is received by the controller and calculates the the response time of the server. The client request from the server is done in a form of a simulation, once the controller generates the data packets and the controller IP will be acting as the destination address. The flow table gets a new entry in a form of the Packet_in message, therefore this message need to be sent by the server to the controller. Further, the controller receives the incoming time of the parsed packet_in message data packet from the respective server. This results, to obtain the response time of the respective server and also the database will be update accordingly.

Step4: Again and again go through **step1, step2, and step3** until end condition is reached**.**

7. Results and Discussion

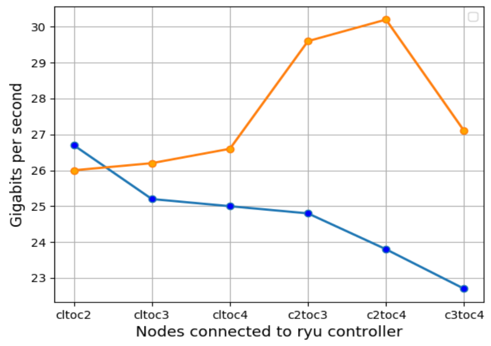


Figure 7.1: Comparison of the throughput between various clients based on Performance metrics

Table 7.1: Performance of the Network based on the throughput

Time	Clients Connected	Throughput1 (Gbps)	Throughput2 (Gbps)
1	C1-C2	26.7	26
2	C1-C3	25.2	26.2
3	C1-C4	25	26.6
4	C2-C3	24.8	29.6
5	C2-C4	23.8	30.2
6	C3-C4	22.7	27.1

The throughput is nothing but the measure of the performance of the network. The comparison of the various clients in the SDN network is done and graph is plotted as shown in figure 7.1 and indicated in the Table 7.1.

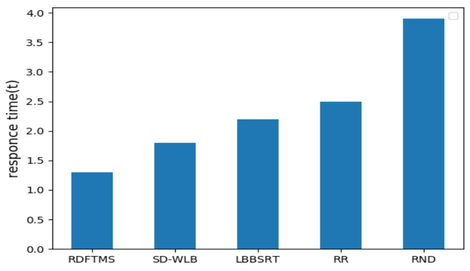


Figure 7.2: Comparison of Average Server Response Time with other methods

From the figure 7.2 and the data shown in table 7.2 we can see that the calculated server response time of the various methods. Each algorithm is executed four times, as shown in the figure 7.2, the proposed algorithm has a lower response time than SD-WLB, LBBSRT, round robin and random algorithms. Based on these methods, we can clear see that the proposed (RDFTMS) method is having less server response time i.e 1.3 seconds with that of RND, RR, LBBSRT, SD-WLB who have 3.9 seconds, 2.5 seconds, 2.2 seconds and 1.8 seconds respectively.

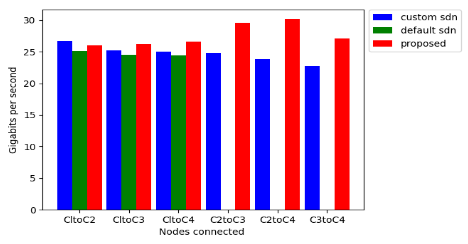


Figure 7.3: Comparison of the TCP Bandwidth of the proposed with that of custom and default topology

As shown in the figure 7.3, we need to calculate the TCP bandwidth between the clients. We have three topologies mentioned here namely, custom SDN, default SDN and the proposed (RDFTMS) SDN. The graph is plotted between Nodes connected vs Gigabits per second. The respective output shows that the suggested methodology has better TCP bandwidth compared to the other two approaches.

The following Table 7.3 signifies the TCP bandwidth between various clients :

Table 7.3: Correlation of the TCP Bandwidth of the proposed , default and Custom SDN Topology

Sl No	Clients Connected	Default SDN Bandwidth (Gbps)	Custom SDN Bandwidth (Gbps)	Proposed SDN Bandwidth (Gbps)
1	C1-C2	25.1	26.7	26.0
2	C1-C3	24.5	25.2	26.2
3	C1-C4	24.4	25	26.6
4	C2-C3	-	24.8	29.6
5	C2-C4	-	23.8	30.2
6	C3-C4	-	22.7	27.1

We need to calculate the TCP Bandwidth between the clients to check the performance of the network. From the Table 7.3 and Figure 7.3 we can see the Comparison of the SDN bandwidth between Custom SDN, Default SDN and the proposed SDN.

Table 7.2: Comparison of the Server Response Time with Various Methods

Sl No	Method Used	Server Response Time (Seconds)
1	RND (Random)	3.9
2	RR(Round Robin)	2.5
3	LBBSRT	2.2
4	SD-WLB	1.8
5	RDFTMS	1.3

The bandwidth of the network is defined as the computation that indicates the maximal capacity of communication media links to transfer the data packets over a network in a particular interval of time. Figure 7.3 explains the TCP bandwidth compared with that of proposed topology, default topology and custom topology.

In the process of calculating the bandwidth using the Open Flow switch with RYU controller, the performance metrics obtained represent the data packets transferred from Client 1(C1) to Client 2 (C2) in the proposed topology is **26.0 Gbps**, in custom topology it is **26.7 Gbps** and in default topology it is **25.1 Gbps**. Similarly for the packets transferred from C1 to C3 in proposed topology it is **26.2 Gbps** and that of custom and default topology it is **25.2 Gbps** and **24.5 Gbps**, between C1 to C4 , for proposed topology it is **26.6 Gbps** and that of custom and default topology it will be **25 Gbps** and **24.4 Gbps**.

The total number of combination of nodes in default topology are three (C1-C2, C1-C3, C1-C4) whereas the proposed topology consists 6 combinations i.e (C1-C2, C1-C3, C1-C4, C2-C3, C2-C4, C3-C4). Additional combinations C2-C3, C2-C4, C3-C4 has no value in default topology. According to the results of bandwidth , we have good improvement in the proposed topology compared to the default topology.

Similarly the proposed topology has good improvement compared to the custom topology as depicted in the table 7.3. The details are mentioned in the table 7.3. From the figure 7.3, we can observe the minimum and maximum throughput of the proposed, custom and default topology in Gbps. The arrival of data packets is the important factor to get high and good performance of SDN network. In the default topology, the minimum and maximum throughput are **24.4 Gbps** and **25.1 Gbps** respectively. As in custom topology, the minimum and maximum throughput are **22.7 Gbps** and **26.7 Gbps** and as far as the proposed topology, the minimum and maximum throughput are **26.0 Gbps** and **30.2 Gbps** respectively.

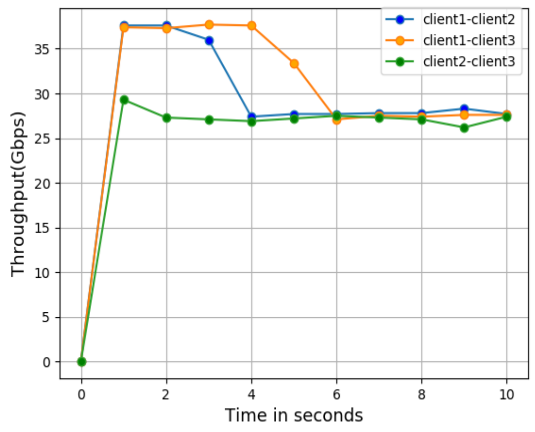


Figure 7.4: Calculation of the Average throughput between different clients in RLFTMS TCP throughput for node to node path

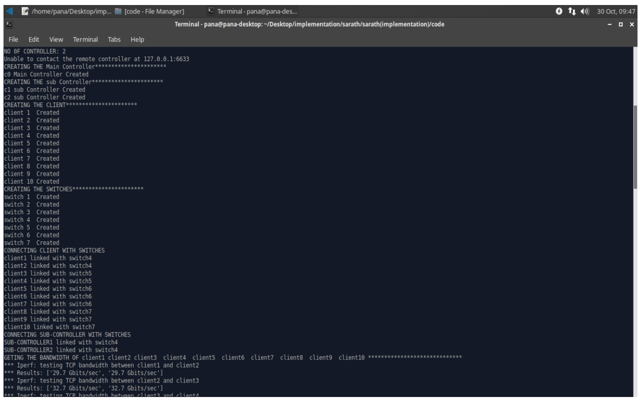
The throughput test in the controller is evaluated for the maximum amount of data, it is processed in a second between two nodes, measured in bits per second or data packets per second. Here iperf3 utility is used to test the controller throughput performance. In order to measure the TCP throughput, iperf3 is executed for 10 seconds on client side and data is collected every 1 second on the server side. The executed result is shown in table 6 and represented graphically in figure 8. It supports to understand about the end-to-end performance of the network.

As in Figure 7.4, highest and lowest throughput in gigabits is approximately **37.6 gigabits** and **27.4 gigabits** between C1 and C2 node, **37.7 gigabits** and **27.1 gigabits** between C1 and C3 node, **29.3 gigabits** and **26.2 gigabits** between C2 and C3 node.

From the Figure 7.4 and Table 7.4, we can see the comparison of the average throughput between different clients i.e Client1-Client2, Client1-Client3 and Client2-Client3. From the result we can clearly mention that the average throughput between Client2-Client3 is better compared to the other comparison of the clients. (iperf3 command used to test the throughput between the clients)

Table 7.4: Comparison of the throughput between the three clients C1, C2 and C3 TCP throughput for three node to node path

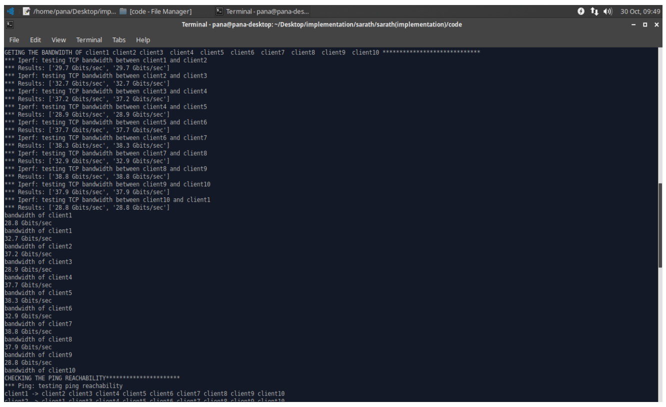
Time (seconds)	C1toC2 Throughput (Gbps)	C1toC3 Throughput (Gbps)	C2toC3 Throughput (Gbps)
0	0	0	0
1	**37.6**	37.4	**29.3**
2	37.6	37.3	27.3
3	36	**37.7**	27.1
4	**27.4**	37.6	26.9
5	27.7	33.4	27.2
6	27.7	**27.1**	27.5
7	27.8	27.5	27.3
8	27.8	27.4	27.1
9	28.3	27.6	**26.2**
10	27.7	27.6	27.4


/home/pana/Desktop/imp...
[code - File Manager]
Terminal - pana@pana-des...
30 Oct, 09:47
Terminal - pana@pana-desktop: ~/Desktop/implementation/sarath/sarath(implementation)/code
File
Edit
View
Terminal
Tabs
Help

```
NO OF CONTROLLER: 2
Unable to contact the remote controller at 127.0.0.1:6633
CREATING THE Main Controller**********************
c0 Main Controller Created
CREATING THE sub Controller**********************
c1 sub Controller Created
c2 sub Controller Created
CREATING THE CLIENT**********************
client 1  Created
client 2  Created
client 3  Created
client 4  Created
client 5  Created
client 6  Created
client 7  Created
client 8  Created
client 9  Created
client 10 Created
CREATING THE SWITCHES**********************
switch 1  Created
switch 2  Created
switch 3  Created
switch 4  Created
switch 5  Created
switch 6  Created
switch 7  Created
CONNECTING CLIENT WITH SWITCHES
client1 linked with switch4
client2 linked with switch4
client3 linked with switch5
client4 linked with switch5
client5 linked with switch6
client6 linked with switch6
client7 linked with switch6
client8 linked with switch7
client9 linked with switch7
client10 linked with switch7
CONNECTING SUB-CONTROLLER WITH SWITCHES
SUB-CONTROLLER1 linked with switch4
SUB-CONTROLLER2 linked with switch4
GETING THE BANDWIDTH OF client1 client2 client3  client4  client5  client6  client7  client8  client9  client10 *****************************
*** Iperf: testing TCP bandwidth between client1 and client2
*** Results: ['29.7 Gbits/sec', '29.7 Gbits/sec']
*** Iperf: testing TCP bandwidth between client2 and client3
*** Results: ['32.7 Gbits/sec', '32.7 Gbits/sec']
*** Iperf: testing TCP bandwidth between client3 and client4
```

Figure 7.5: Connection Establishment between the Controller and the Switches

Initially, connection should be established between the controller and switches. Here the main controller used is RYU. This controller is connected to switches via subcontrollers. It is as shown in the figure 7.5.

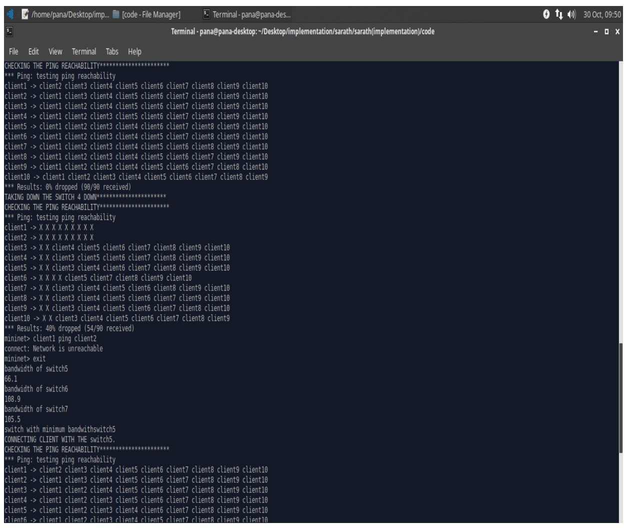

/home/pana/Desktop/imp...
[code - File Manager]
Terminal - pana@pana-des...
30 Oct, 09:49
Terminal - pana@pana-desktop: ~/Desktop/implementation/sarath/sarath(implementation)/code
File
Edit
View
Terminal
Tabs
Help

```
GETING THE BANDWIDTH OF client1 client2 client3  client4  client5  client6  client7  client8  client9  client10 *****************************
*** Iperf: testing TCP bandwidth between client1 and client2
*** Results: ['29.7 Gbits/sec', '29.7 Gbits/sec']
*** Iperf: testing TCP bandwidth between client2 and client3
*** Results: ['32.7 Gbits/sec', '32.7 Gbits/sec']
*** Iperf: testing TCP bandwidth between client3 and client4
*** Results: ['37.2 Gbits/sec', '37.2 Gbits/sec']
*** Iperf: testing TCP bandwidth between client4 and client5
*** Results: ['28.9 Gbits/sec', '28.9 Gbits/sec']
*** Iperf: testing TCP bandwidth between client5 and client6
*** Results: ['37.7 Gbits/sec', '37.7 Gbits/sec']
*** Iperf: testing TCP bandwidth between client6 and client7
*** Results: ['38.3 Gbits/sec', '38.3 Gbits/sec']
*** Iperf: testing TCP bandwidth between client7 and client8
*** Results: ['32.9 Gbits/sec', '32.9 Gbits/sec']
*** Iperf: testing TCP bandwidth between client8 and client9
*** Results: ['38.8 Gbits/sec', '38.8 Gbits/sec']
*** Iperf: testing TCP bandwidth between client9 and client10
*** Results: ['37.9 Gbits/sec', '37.9 Gbits/sec']
*** Iperf: testing TCP bandwidth between client10 and client1
*** Results: ['28.8 Gbits/sec', '28.8 Gbits/sec']
bandwidth of client1
28.8 Gbits/sec
bandwidth of client1
32.7 Gbits/sec
bandwidth of client2
37.2 Gbits/sec
bandwidth of client3
28.9 Gbits/sec
bandwidth of client4
37.7 Gbits/sec
bandwidth of client5
38.3 Gbits/sec
bandwidth of client6
32.9 Gbits/sec
bandwidth of client7
38.8 Gbits/sec
bandwidth of client8
37.9 Gbits/sec
bandwidth of client9
28.8 Gbits/sec
bandwidth of client10
CHECKING THE PING REACHABILITY**********************
*** Ping: testing ping reachability
client1 -> client2 client3 client4 client5 client6 client7 client8 client9 client10
```

Figure 7.6: Calculation of Bandwidth between the client using iperf command

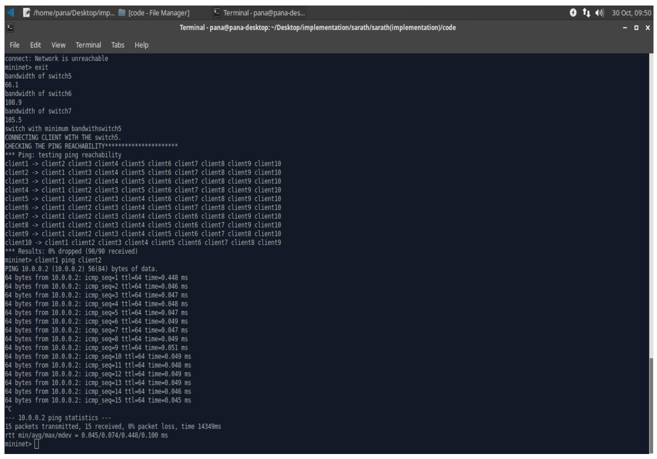
Once, the connection is established, using the iperf command, calculation of Bandwidth in switches are done. The minimum bandwidth is taken among them and it is used for the further process. It is as shown in the figure 7.6.

To check whether the connection is established properly, we need to conduct the ping test between the client and server. This process is carried out as shown in the figure 7.7.


/home/pana/Desktop/imp...
[code - File Manager]
Terminal - pana@pana-des...
30 Oct, 09:50
Terminal - pana@pana-desktop: ~/Desktop/implementation/sarath/sarath(implementation)/code
File
Edit
View
Terminal
Tabs
Help

```
CHECKING THE PING REACHABILITY**********************
*** Ping: testing ping reachability
client1 -> client2 client3 client4 client5 client6 client7 client8 client9 client10
client2 -> client1 client3 client4 client5 client6 client7 client8 client9 client10
client3 -> client1 client2 client4 client5 client6 client7 client8 client9 client10
client4 -> client1 client2 client3 client5 client6 client7 client8 client9 client10
client5 -> client1 client2 client3 client4 client6 client7 client8 client9 client10
client6 -> client1 client2 client3 client4 client5 client7 client8 client9 client10
client7 -> client1 client2 client3 client4 client5 client6 client8 client9 client10
client8 -> client1 client2 client3 client4 client5 client6 client7 client9 client10
client9 -> client1 client2 client3 client4 client5 client6 client7 client8 client10
client10 -> client1 client2 client3 client4 client5 client6 client7 client8 client9
*** Results: 0% dropped (90/90 received)
TAKING DOWN THE SWITCH 4 DOWN**********************
CHECKING THE PING REACHABILITY**********************
*** Ping: testing ping reachability
client1 -> X X X X X X X X X
client2 -> X X X X X X X X X
client3 -> X X client4 client5 client6 client7 client8 client9 client10
client4 -> X X client3 client5 client6 client7 client8 client9 client10
client5 -> X X client3 client4 client6 client7 client8 client9 client10
client6 -> X X X X client5 client7 client8 client9 client10
client7 -> X X client3 client4 client5 client6 client8 client9 client10
client8 -> X X client3 client4 client5 client6 client7 client9 client10
client9 -> X X client3 client4 client5 client6 client7 client8 client10
client10 -> X X client3 client4 client5 client6 client7 client8 client9
*** Results: 40% dropped (54/90 received)
mininet> client1 ping client2
connect: Network is unreachable
mininet> exit
bandwidth of switch5
66.1
bandwidth of switch6
108.9
bandwidth of switch7
105.5
switch with minimum bandwithswitch5
CONNECTING CLIENT WITH THE switch5.
CHECKING THE PING REACHABILITY**********************
*** Ping: testing ping reachability
client1 -> client2 client3 client4 client5 client6 client7 client8 client9 client10
client2 -> client1 client3 client4 client5 client6 client7 client8 client9 client10
client3 -> client1 client2 client4 client5 client6 client7 client8 client9 client10
client4 -> client1 client2 client3 client5 client6 client7 client8 client9 client10
client5 -> client1 client2 client3 client4 client6 client7 client8 client9 client10
client6 -> client1 client2 client3 client4 client5 client7 client8 client9 client10
```

Figure 7.7: Ping Test Connectivity between the client and the server to check network reachability


/home/pana/Desktop/imp...
[code - File Manager]
Terminal - pana@pana-des...
30 Oct, 09:50
Terminal - pana@pana-desktop: ~/Desktop/implementation/sarath/sarath(implementation)/code
File
Edit
View
Terminal
Tabs
Help

```
connect: Network is unreachable
mininet> exit
bandwidth of switch5
66.1
bandwidth of switch6
108.9
bandwidth of switch7
105.5
switch with minimum bandwithswitch5
CONNECTING CLIENT WITH THE switch5.
CHECKING THE PING REACHABILITY**********************
*** Ping: testing ping reachability
client1 -> client2 client3 client4 client5 client6 client7 client8 client9 client10
client2 -> client1 client3 client4 client5 client6 client7 client8 client9 client10
client3 -> client1 client2 client4 client5 client6 client7 client8 client9 client10
client4 -> client1 client2 client3 client5 client6 client7 client8 client9 client10
client5 -> client1 client2 client3 client4 client6 client7 client8 client9 client10
client6 -> client1 client2 client3 client4 client5 client7 client8 client9 client10
client7 -> client1 client2 client3 client4 client5 client6 client8 client9 client10
client8 -> client1 client2 client3 client4 client5 client6 client7 client9 client10
client9 -> client1 client2 client3 client4 client5 client6 client7 client8 client10
client10 -> client1 client2 client3 client4 client5 client6 client7 client8 client9
*** Results: 0% dropped (90/90 received)
mininet> client1 ping client2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.448 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.046 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.047 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.048 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.047 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.049 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.047 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.049 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.051 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=0.049 ms
64 bytes from 10.0.0.2: icmp_seq=11 ttl=64 time=0.048 ms
64 bytes from 10.0.0.2: icmp_seq=12 ttl=64 time=0.049 ms
64 bytes from 10.0.0.2: icmp_seq=13 ttl=64 time=0.049 ms
64 bytes from 10.0.0.2: icmp_seq=14 ttl=64 time=0.046 ms
64 bytes from 10.0.0.2: icmp_seq=15 ttl=64 time=0.045 ms
^C
--- 10.0.0.2 ping statistics ---
15 packets transmitted, 15 received, 0% packet loss, time 14349ms
rtt min/avg/max/mdev = 0.045/0.074/0.448/0.100 ms
mininet> 
```

Figure 7.8: Calculation of Round Trip Time using Ping Command between Client and Server.

As shown in the figure 7.8, the round trip time between the client and the server is calculated after performing the ping command and clearly showing the number of packets transmitted, received and also any packet loss has occurred and even the time to obtain the round trip time.

Based on the results obtained and comparison done with that of the traditional methods such as **SD-WLB, LBBSRT, Round Robin (RR)** and **Random selection** methods, the **RDFTMS** mechanism is having superiority and there is improvement w.r.t the average server response time by **18%, 19.58%, 33.94%** and **57.41%** respectively. Similarly, the enhancement of throughput in an average w.r.t these algorithmic techniques are also considered and they appear to be **8.25%, 16.52%, 29.72% and 58.27%** respectively.

8. Conclusion

Load balancing based on the server response time, throughput is being compared , this improves the server workload ditribution among various datacenters. Ultimately the customers or clients will have a great experience in terms of the reducing response time and increase in the throughput. The controller is considered as the master mind of the network and is majorly anlayzed to get the flow of network traffic in the real time so that it is easily monitored to get the clear idea of the packet flow in the heavy network traffic. For any efficient client to server communication to happen, i.e nothing but the smooth traffic movement between any two nodes , we need an effective and efficient mechansim of movement of data in the form of packets so that easy flow of data can happen without any transmission impairment. In this work the controller used is RYU. It is mainly used to gain resource utilization so that the network traffic is better and to achieve high network performance. In future, using any deep learning or machine learning mechanisms with other SDN controllers, calculate the performance of the network in terms of server response time, throughput by considering other metrics like packet loss, delay, jitter. Finally developing a improvised, reliable, fault tolerant SDN for the efficient communication between the intended client or the host and the respective server.

Acknowledgements.

This is to acknowledge the Department of Computer Science and Engineering, Puducherry Technological University for giving the opportunity to conduct the experiments and utlization of the research lab to carry out the work. Further, I would thank all the faculty and staff , Puducherry Technological University for kind cooperation for carrying out this work successfully.

References

[1.] Soleimanzadeh, K., Ahmadi, M. and Nassiri, M., 2019. SD-WLB: An SDN-aided mechanism for web load balancing based on server statistics. *ETRI Journal*, *41*(2), pp.197-206.

[2.] Zhong, H., Lin, Q., Cui, J., Shi, R. and Liu, L., 2015. An efficient SDN load balancing scheme based on variance analysis for massive mobile users. *Mobile Information Systems*, *2015*.

[3.] Kaur, S., Kumar, K., Singh, J. and Ghumman, N.S., 2015, March. Round-robin based load balancing in Software Defined Networking. In *2015 2nd international conference on computing for sustainable global development (INDIACom)* (pp. 2136- 2139). IEEE.

[4.] Kang, B. and Choo, H., 2018. An SDN-enhanced load-balancing technique in the cloud system. *The Journal of Supercomputing*, *74*(11), pp.5706-5729.

[5.] Linn, A.S., Win, S.H. and Win, S.T., 2019. Server Load Balancing in Software Defined Networking.

[6.] Hamed, M.I., ElHalawany, B.M., Fouda, M.M. and Tag Eldien, A.S., 2017. Performance analysis of applying load balancing strategies on different SDN environments. *Benha Journal of Applied Sciences*, *2*(1), pp.1-7.

[7.]Islam, M.T., Islam, N. and Al Refat, M., 2020. Node to node performance evaluation through RYU SDN controller. *Wireless Personal Communications*, pp.1-16.

[8.] Bhardwaj, S. and Panda, S.N., 2022. Performance Evaluation Using RYU SDN Controller in Software-Defined Networking Environment. *Wireless Personal Communications*, *122*(1), pp.701-723.

[9.] Silva, J.B., Silva, F.S.D., Neto, E.P., Lemos, M. and Neto, A., 2020. Benchmarking of mainstream SDN controllers over open off-the-shelf software-switches. *Internet Technology Letters*, *3*(3), p.e152.

[10.] CN, S., 2019. A proactive flow admission and re-routing scheme for load balancing and mitigation of congestion propagation in SDN data plane. *International Journal of Computer Networks & Communications (IJCNC) Vol*, *10*.

[11.] Kiadehi, K.B., Rahmani, A.M. and Molahosseini, A.S., 2021. Increasing fault tolerance of data plane on the internet of things using the software-defined networks. *PeerJ Computer Science*, *7*, p.e543.

[12.]Xin-gang, W., 2018, March. A Link Performance-based Failure Recovery Approach in SDN Data Plane. In *Proceedings of the 3rd International Conference on Multimedia and Image Processing* (pp. 46-51).

[13.] Rehman, A.U., Aguiar, R.L. and Barraca, J.P., 2019. Fault-tolerance in the scope of software-defined networking (sdn). *IEEE Access*, *7*, pp.124474-124490.

[14.] Yamansavascilar, B., Baktir, A.C., Ozgovde, A. and Ersoy, C., 2020. Fault tolerance in SDN data plane considering network and application based metrics. *Journal of Network and Computer Applications*, *170*, p.102780.

[15.] Karakus, M. and Durresi, A., 2017. A survey: Control plane scalability issues and approaches in software-defined networking (SDN). *Computer Networks*, *112*, pp.279-293.

[16.] Das, R.K., Pohrmen, F.H., Maji, A.K. and Saha, G., 2020. FT-SDN: a fault-tolerant distributed architecture for software defined network. *Wireless personal communications*, *114*(2), pp.1045-1066.

[17.] Seidlitz, L. and Perner, C., 2020. Fault tolerance in SDN. *Network*, *45*.

[18.] Liang, D., Liu, Q., Yan, B., Hu, Y., Zhao, B. and Hu, T., 2021. Low interruption ratio link fault recovery scheme for data plane in software-defined networks. *Peer-to-Peer Networking and Applications*, *14*(6), pp.3806-3819.

[19.] Dolynnyi, O., Nikolskiy, S. and Kulakov, Y., 2020. THE METHOD OF SDN CLUSTERING FOR CONTROLLER LOAD BALANCING. *Information, Computing and Intelligent systems*, (1).

[20.] Das, T. and Gurusamy, M., 2020. Controller placement for resilient network state synchronization in multi-controller sdn. *IEEE Communications Letters*, *24*(6), pp.1299-1303.

[21.] Zhou, Z., Benson, T.A., Canini, M. and Chandrasekaran, B., 2021, October. Tardis: A Fault-Tolerant Design for Network Control Planes. In *Proceedings of the ACM SIGCOMM Symposium on SDN Research (SOSR)* (pp. 108-121).

[22.] Shailly, M., 2021. A critical review based on Fault Tolerance in Software Defined Networks. *Turkish Journal of Computer and Mathematics Education (TURCOMAT)*, *12*(2), pp.456-461.

[23.] Rajoriya, M.K. and Gupta, C.P., 2021, April. A Taxonomy on Distributed Controllers in Software Defined Networking. In *2021 5th International Conference on Computing Methodologies and Communication (ICCMC)* (pp. 120-126). IEEE.

[24.] Ahmad, S. and Mir, A.H., 2021. Scalability, consistency, reliability and security in sdn controllers: A survey of diverse sdn controllers. *Journal of Network and Systems Management*, *29*(1), pp.1-59.

[25.] Sminesh, C.N., Kanaga, E.G.M. and Sreejish, A.G., 2020. A multi-controller placement strategy in software defined networks using affinity propagation. *International Journal of Internet Technology and Secured Transactions*, *10*(1-2), pp.229-253.

[26.] Alenazi, M.J. and Cetinkaya, E.K., 2020. Resilient placement of SDN controllers exploiting disjoint paths. *Transactions on Emerging Telecommunications Technologies*, *31*(2), p.e3725.

[27.] Ruaro, M. and Moraes, F.G., 2020, August. Multiple-objective Management based on a Distributed SDN Architecture for Many-cores. In *2020 33rd Symposium on Integrated Circuits and Systems Design (SBCCI)* (pp. 1-6). IEEE.

[28.] Phemius, K., Bouet, M. and Leguay, J., 2014, May. Disco: Distributed multi-domain sdn controllers. In *2014 IEEE Network Operations and Management Symposium (NOMS)* (pp. 1-4). IEEE.

[29.] Yang, S., Cui, L., Chen, Z. and Xiao, W., 2020. An efficient approach to robust SDN controller placement for security. *IEEE Transactions on Network and Service Management*, *17*(3), pp.1669-1682.

[30.] Satheesh, N., Rathnamma, M.V., Rajeshkumar, G., Sagar, P.V., Dadheech, P., Dogiwal, S.R., Velayutham, P. and Sengan, S., 2020. Flow-based anomaly intrusion detection using machine learning model with software defined networking for OpenFlow network. *Microprocessors and Microsystems*, *79*, p.103285.

[31.] Rai, P. and Sarma, H.K.D., 2022. Reliable Data Delivery in Software-Defined Networking: A Survey. In *Contemporary Issues in Communication, Cloud and Big Data Analytics* (pp. 3-17). Springer, Singapore.