

Evaluation of Routing Performance using OSPF and Multi-Controller Based Network Architecture

Nicholas. J. Omumbo

Maseno University, School of Computing and informatics, Kisumu, Kenya
E-mail: nicholas.j.omumbo@gmail.com

Titus. M. Muhambe and Cyprian M. Ratemo

Maseno University, School of Computing and informatics, Kisumu, Kenya
Kisii University, School of Computing and informatics Kenya
E-mail: {muhambemukisa, cratemo}@gmail.com

Received: 27 November 2020; Accepted: 28 March 2021; Published: 08 August 2021

Abstract: Newer mobile applications are increasingly being defined using Internet Protocol, resulting in increased use of Internet Protocol and subsequent upsurge of smartphones. However, many communication service provider core networks continue to use classical routing protocols and single controller-based networks if deployed. Controller-based networks built on the foundation of software-defined networks include centralization and separation of control plane and data plane, which can address the challenges experienced with the classical routing protocols. When single controllers are used, they tend to get overloaded with traffic. The ability to use multi-controller-based network architecture to improve quality of service in the mobile IP core network is still an open issue. This paper presents a performance evaluation of multi-controller-based network architecture, running OpenFlow and Open Shortest Path First protocol. The long-term evolution simulated network architecture is created using well-known network simulator Objective Modular Network Testbed running OpenFlow and simuLTE add-on. We test and analyze data traffic for Packet data ratio and Jitter and their associated effects on a multi-controller-based network running OpenFlow versus OSPF on a mobile core network. The experiment created two topologies; multi controller-based and Open Shortest path first network. Video and ping traffic is tested by the generation of traffic from User Equipment to the network-based server in the data center and back, and traffic metrics recorded on an inbuilt integrated development environment. The simulation setup consisted of an OpenFlow controller, HyperFlow algorithm, OpenFlow switches, and Open Shortest Path First routers. The multi-controller-based network improved Jitter by 10 ms. The Open Shortest Path first showed packet data ratio values of 89% gain while the controller-based network registered a value of 86%. A standard deviation test revealed 0.7%, which shows that the difference is not significant when testing for Packet data ratio. We provided insight into the performance of multi-controller-based architecture and Open Shortest Path First protocol in the communication service provider's core network.

Index Terms: OpenFlow, Open Shortest Path First, Multi-controller, Quality of Service, HyperFlow, OMNeT++, LTE.

1. Introduction

Increased ownership of handheld mobile devices has exceeded expectations, coupled with the upward growth of new applications being defined using internet protocol (IP) [1–6]]. Studies by [7, 8] and [9] show that mobile IP networks still use classical routing protocols in the core. Such protocols lack the programmability and scalability desired of routing in mobile IP networks, as shown by [10–13]. According to [14] and [15], classical networks has only part of the network information, and this behavior can lead to incorrect routing decisions and subsequent cause of loops in the network. Controller-based networks, an instance of software-defined network (SDN), is a new network paradigm that started gaining prominence in 2010 [10, 11, 16, 17] and whose strength is based on centrality and programmability of the network routing process [18]. As a new networking norm, controller-based network architecture promises to address routing deficiencies posed by classical mobile IP core network routing.

The need to orchestrate and automate routing in mobile IP networks operation from a simplified central point coupled with an increased demand for multiple controllers in a network is a clear motivation to explore and establish the degree to which multiple controller-based networks can provide redundancy and at the same time improve QoS metrics in mobile IP network core [19, 20]. Furthermore, controller-based networks have entrenched themselves alongside network function virtualization (NFV), cloud, and intent-based networking (IBN) as next-generation routing solutions for mobile IP core networks [4, 10, 21–23].

Controller-based networks can improve QoS performance in wired networks, as demonstrated by [24]. Moreover, multiple controllers have primarily been used to enhance the network's high availability, as shown by [25]. The ability to use multiple controller architecture to improve QoS routing performance in mobile network IP is still an issue, as established by [19, 20].

Therefore, in this paper, we examine the use of multiple controllers with OpenFlow and OSPF to establish how the two protocols can affect PDR and Jitter in a mobile IP network with a multipath destination core.

This study modeled an OSPF and multiple controller-based networks running OpenFlow using simulator OMNeT++, OpenFlow, and simuLTE add-ons [26, 27]. The focus was on two QoS parameters, packet delivery ratio (PDR) and Jitter, which have been identified by [2] as essential measures to note when looking at the quality of experience (QoE) in IP communication service provider network (CSP). This study provides insight into how multiple controllers can improve QoS in the CSP network compared to traditional network architecture running OSPF.

In this paper's remainder, we present as follows; in section 2, we provide related work on controllers, multiple controller architecture, QoS measurement, QoS baseline, and classical routing protocol, specifically open shortest path first (OSPF). In section 3, we outline the methodology used, and in section 4, we provide the test environment and simulation exercise. In section 5, we discuss the simulation results. Finally, in section 6, we give a conclusion and brief recommendation for future work.

2. Related Work

2.1. Multiple controller-based network architecture

According to [28, 29], there is a change where the control plane resides in a controller-based architecture. The controller performs the logic function of the network while operating from a centralized platform. Due to their centralized nature, controllers provide an abstraction of hardware and gives the controller a familiar interface referred to as application programming interface (API) to communicate with network devices like switches, routers, firewalls, and load balancers, amongst others [12, 30]. Also, the abstraction allows for easier programmability of the network. Controllers use the northbound interface (NBI) and southbound interface (SBI) to communicate between other controllers and network devices.

Northbound interface (NBI) avails the controller so that other programs can use its data and functions. SBI allows for communication between the controller and other network devices. More theoretical details on SBI, NBI, and API are illustrated in [25, 31]. Fig. 1 describes how SBI and NBI are interlinked in controller-based network architecture.

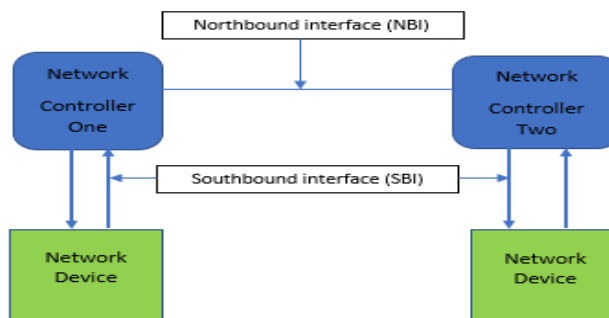


Fig.1. Southbound and Northbound Interfaces

Initially, controller implementation tended to implement single controllers to operate in the network with the primary goal of simplicity, which is demonstrated by [19, 20, 32]. Single controllers had two critical deficiencies: the controller's traffic increases considerably, especially with new switches in the network, and the potential for an increased end-to-end delay if the network has a longer length [19, 33]. Fig.2. demonstrates multiple controllers with flows going to only the relevant controller in Fig.3. The controller's operation is coordinated via an application inbuilt in OMNeT++ OpenFlow known as HyperFlow, which supports load-balancing. HyperFlow detailed operation is explained in section 3. In contrast, in Fig.3, we have a single controller architecture with almost all flows directed at a single controller.

In later deployments, multi-controllers were deployed to address high availability challenges resulting from single controllers [19, 33].

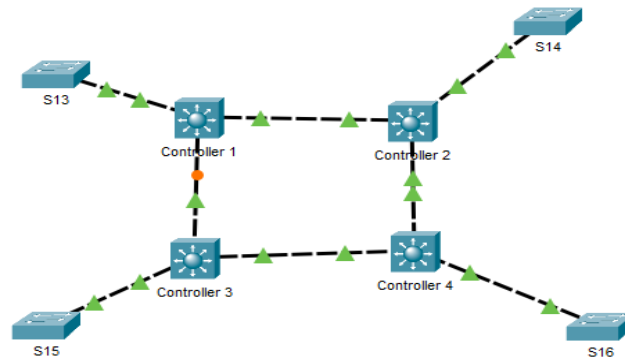


Fig.2. Multiple controller architecture depicting fewer flows to anyone controller

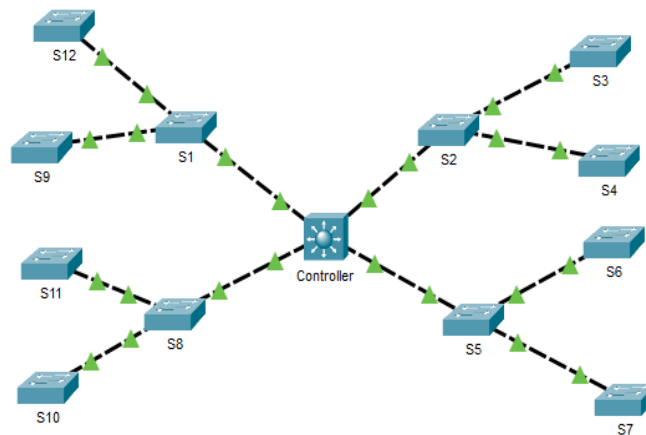


Fig.3. Single controller switch with all flows sent towards a single controller

In multiple controller architecture, a set of controllers that work together to achieve performance and scalability is deployed [10, 19]. While there are many models of multiple controller architecture as illustrated by [19, 25, 34] in this paper, we limit ourselves to the centralized physical architecture as part of the OMNeT++ simulation model, which we explain in the next paragraph.

As the name suggests, centralized physical architecture places multiple controllers in a central location and connects them via a load balancing protocol known as HyperFlow [25]. The load balancing protocol can automate load across two controllers or more depending on configuration, and end-user needs are further explained in [25, 35, 36].

In [36], they reviewed controllers to establish a single controller's efficacy versus multiple controllers. It was reported that there was a better performance concerning the use of multiple controllers. While studies [19, 36] acknowledge that single controllers have deficiencies, they mostly focus on redundancy aspects of multi-controller-based networks. In [35], they provided a theoretical review about SDN and the use of multi-controllers with an emphasis on high availability.

A literature review does not reveal any comparison studies on QoS performance levels of OSPF and multiple controller-based architectures in the mobile IP network.

2.2. Open Shortest Path First: An Overview

Open shortest path first (OSPF) is a well-known open-source link-state protocol [37–39]. In this paper, OSPF is classified as a classical routing protocol principally because of its distributed operation nature. OSPF was selected for this experiment due to its vast deployment and readily available OSPF-enabled routers in the OMNeT++ INET framework. Details on the operation, deployment, and configuration of OSPF protocol can be found [37, 40].

2.3. Quality of Service Metrics and Baseline

Quality of service constitutes an essential feature of CSP network infrastructure as it also affects routing speeds [2]. Most regulatory bodies spell out QoS measures that are expected of CSP companies [2, 41]. This paper adopts the QoS measures for PDR and Jitter, as illustrated in Table 1.

Table 1. Quality of Service Baseline

Application	Jitter	PDR
Video	<=25ms	>=95%
Ping	30-50ms	0.1%

The QoS metrics in consideration were PDR and Jitter. As observed earlier in section 2, lower Jitter and higher PDR are desirable for mobile IP networks. In (1), PDR is calculated as a number of received packets over the number of generated packets multiplied by 100 percent. Jitter is the delay between consecutive packets and is calculated in (2) as follows.

$$PDR = \text{sentpackets} / \text{receivedpackets} * 100 \tag{1}$$

$$Jitter = D(i) - D(i-1) \text{ where } D = \text{Forwarding Delay and } i = \text{order of packets arrival} \tag{2}$$

3. Deploying Multi-Controller and OSPF Network Architecture

3.1. Simulation Network Design; Controller based versus OSPF

We extended the all IP network (AIPN) in [26], built with simuLTE, OpenFlow plugins, and running on OMNeT++ emulator. For purposes of this experiment, we designed and deployed a Multi-controller and OSPF routed architecture. We used commonly deployed hybrid topology consisting of mesh and star topology in the CSP core network [37]. Later in section three, we outline the two topological structures used to conduct this experiment. We leveraged the two OpenFlow controller’s ability to improve routing QoS through load balancing of IP traffic for any destination. OpenFlow controller uses expanded routing table parameters such as Terms of Service (ToS) header, ingress interface, vlan address, source, and destination IP to forward traffic with multiple actions such as modifying, dropping, forward, or queuing, etc. [28, 42, 43]. The expanded routing parameters directly contrast to OSPF, mainly using source and destination IP with three actions encapsulate, drop, or forward [38]. Fig.4 and Fig. 5 illustrate the operational structure of the OpenFlow Controller and OSPF network.

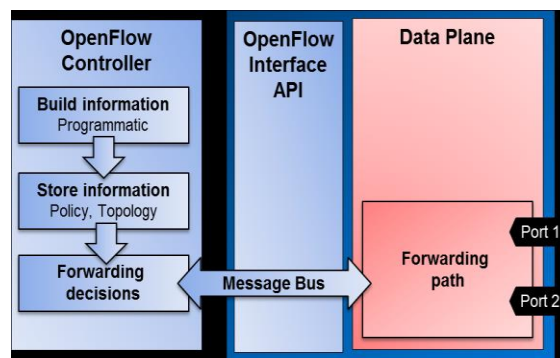


Fig.4. OpenFlow network

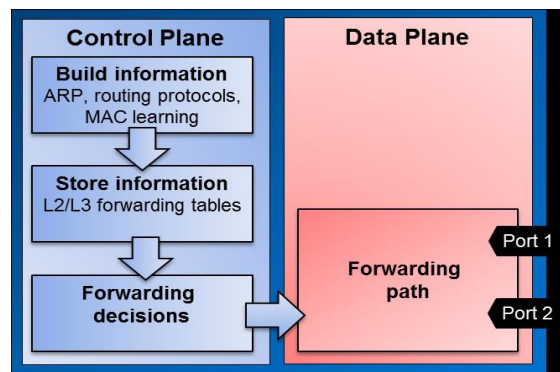


Fig.5. OSPF routed network

We used the HyperFlow protocol to load balance traffic between Controller1 and Controller2 albeit transparently [25]). HyperFlow uses OpenFlow switches as forwarding components, OpenFlow controller. An instance of HyperFlow is enabled on each controller. The controllers have a consistent domain view of the network, and each OpenFlow switch is connected to the best controller in its line of sight (LoS). Extensive details on HyperFlow configuration, deployment, and operation can be found in [25]. The HyperFlow algorithm is illustrated in Appendix 2. In the next subsection, we outline the methodology used to create our proposed AIPN core network extension with the OSPF and Multi-Controller network architectures. The main network architecture consisted of two files (1) Omnet.ini, which has the execution files to run the experiment (2) the network development file (NED) that was used to create the two network topologies; OSPF and Multi-controller OpenFlow routed networks.

3.2. Codes for the CSP network architecture and execution files

Code 1 Omnetpp.ini for launching and running the experiment

```

1. record-eventlog = true
2. **.vector-recording = true
3. **.scalar-recording = true
4. **.statistic-recording = true
5. #####
6. # CSP Core Network Configuration #
7. #####
8. **.pgwStandard.trafficFlowFilter.filterFileName = "lteCoreConfigFiles/pgw.xml"
9. **.pgwStandard.gtp_user.teidFileName = "lteCoreConfigFiles/pgw.xml"
10. **.pgwStandard.gtp_user.tftFileName = "lteCoreConfigFiles/pgw.xml"
11. **.sgwStandard1.gtp_user.teidFileName = "lteCoreConfigFiles/sgw1.xml"
12. ##### Number of Resource Blocks #####
13. **.numRbDL = 6
14. **.numRbUL = 6
15. **.binder.numBands = 6 # this value should be kept equal to the number of RBs
16. ##### Transmission Power #####
17. **.ueTxPower = 26
18. **.eNodeBTxPower = 40
19. # Schedulers
20. **.mac.schedulingDisciplineDL = "MAXCI"
21. **.mac.schedulingDisciplineUL = "MAXCI"
22. **.rtt.result-recording-modes = +vector,-stats
23. **.packets.result-recording-modes = +count
24. **.packetBytes.result-recording-modes = +sum
25. **.packets*.scalar-recording = true
26. **.packetBytes*.scalar-recording = true
27. **.numPacketIn*.scalar-recording = true
28. **.pingApp[*].numLost*.scalar-recording = true
29. **.pingApp[*].numOutOfOrderArrivals*.scalar-recording = true
30. **.pingApp[*].rtt*.vector-recording = true

```

Code for OSPF network

```

1. [Config CSP-OSPF]
2. network = lte.simulations.networks.eutran_epcNetwork2
3. #core network extension
4. **.eNB1.trafficFlowFilter.filterFileName = "lteCoreConfigFiles/enb1.xml"
5. **.eNB1.gtp_user.teidFileName = "lteCoreConfigFiles/enb1.xml"
6. **.eNB1.gtp_user.tftFileName = "lteCoreConfigFiles/enb1.xml"
7. **.eNB2.trafficFlowFilter.filterFileName = "lteCoreConfigFiles/enb1.xml"
8. **.eNB2.gtp_user.teidFileName = "lteCoreConfigFiles/enb2.xml"
9. **.eNB2.gtp_user.tftFileName = "lteCoreConfigFiles/enb2.xml"
10. **.eNB3.trafficFlowFilter.filterFileName = "lteCoreConfigFiles/enb3.xml"
11. **.eNB3.gtp_user.teidFileName = "lteCoreConfigFiles/enb3.xml"
12. **.eNB3.gtp_user.tftFileName = "lteCoreConfigFiles/enb3.xml"
13. **.eNB4.trafficFlowFilter.filterFileName = "lteCoreConfigFiles/enb3.xml"
14. **.eNB4.gtp_user.teidFileName = "lteCoreConfigFiles/enb4.xml"
15. **.eNB4.gtp_user.tftFileName = "lteCoreConfigFiles/enb4.xml"
16. **.eNB5.trafficFlowFilter.filterFileName = "lteCoreConfigFiles/enb5.xml"

```

```

17. **.eNB5.gtp_user.teidFileName = "lteCoreConfigFiles/enb5.xml"
18. **.eNB5.gtp_user.tftFileName = "lteCoreConfigFiles/enb5.xml"
19. # connect each UE to the eNB
20. **.ue1[*].macCellId = 1
21. **.ue1[*].masterId = 1
22. **.ue2[*].macCellId = 2
23. **.ue2[*].masterId = 2
24. **.ue3[*].macCellId = 3
25. **.ue3[*].masterId = 3
26. **.ue4[*].macCellId = 4
27. **.ue4[*].masterId = 4
28. **.ue5[*].macCellId = 5
29. **.ue5[*].masterId = 5
30. #evolved Node B configurations
31. *.eNB*.mobility.initFromDisplayString = false
32. *.eNB*.mobility.initialX = 0m
33. *.eNB*.mobility.initialY = 0m
34. # UE positioning and mobility
35. *.ue*[*].mobility.acceleration = 0
36. *.ue*[*].mobility.angle = uniform(0deg, 360deg)
37. *.ue*[*].mobility.constraintAreaMaxX = 1000m
38. *.ue*[*].mobility.constraintAreaMaxY = 1000m
39. *.ue*[*].mobility.constraintAreaMinX = 0m
40. *.ue*[*].mobility.constraintAreaMinY = 0m
41. *.ue*[*].mobility.initFromDisplayString = false
42. *.ue*[*].mobility.initialX = uniform(0m,300m)
43. *.ue*[*].mobility.initialY = uniform(0m,300m)
44. *.ue*[*].mobility.initialZ = 0
45. *.ue*[*].mobility.speed = 0.5mps
46. *.ue*[*].mobilityType = "LinearMobility"
47. #===== Application 1 Setup =====
48. **.numUe = ${numUes=10,20,30}
49. *.InternetHost1.numUdpApps = ${numUes}
50. # Video Client Configuration
51. **.ue*.udpApp[0].typename = "UDPVideoStreamCli"
52. **.ue*.udpApp[0].serverAddress = "InternetHost1"
53. **.ue*.udpApp[0].localPort = 9999
54. **.ue*.udpApp[0].serverPort = 3088
55. **.ue*.udpApp[0].startTime = 0s
56. # Video Server Configuration
57. *.InternetHost1.udpApp[*].typename = "UDPVideoStreamSvr"
58. *.InternetHost1.udpApp[*].videoSize = 20MiB
59. *.InternetHost1.udpApp[*].localPort = 3088
60. *.InternetHost1.udpApp[*].sendInterval = 10ms
61. *.InternetHost1.udpApp[*].packetLen = 10000B
62. #-----#
#===== OSPF Routing Setup =====
**.ospf.ospfConfig = xmldoc("ASConfig.xml")

```

Code for Multi-Controller network

```

1. Config CSP-M-Controller]
2. network = lte.simulations.networks.eutran_epcNetwork3
3. **.flowTable**.scalar-recording = true
4. **.controllerApps[*].*.scalar-recording = true
5. #Openflow configurations
6. **.DC10.OF_Switch.connectAddress = "controller1"
7. **.DC11.OF_Switch.connectAddress = "controller1"
8. **.DC12.OF_Switch.connectAddress = "controller1"
9. **.DC13.OF_Switch.connectAddress = "controller1"
10. **.DC20.OF_Switch.connectAddress = "controller2"

```

```

11. **.DC21.OF_Switch.connectAddress = "controller2"
12. **.DC22.OF_Switch.connectAddress = "controller2"
13. **.DC23.OF_Switch.connectAddress = "controller2"
14. **.DC24.OF_Switch.connectAddress = "controller2"
15. **.DC*.OF_Switch.connectAt = uniform(0s,1s)
16. #buffer size
17. **.OF_Switch.bufferCapacity = 3712
18. **.OF_Switch.serviceTime = 0.000035s * (${switchServiceTimeFactor=
1,130,140,150,152,154,156,158,160,170,180,190,200})
19. #core speeds
20. **.controller*.serviceTime = 0.000005556s * (${controllerServiceTimeFactor=
1,130,140,150,152,154,156,158,160,170,180,190,200 !switchServiceTimeFactor})
21. #synchronize
22. **.HF*.serviceTime = 0.001013171s* (${synchronizerServiceTimeFactor=
1,130,140,150,152,154,156,158,160,170,180,190,200 !switchServiceTimeFactor})
23. **.controller*.numControllerApps = 2
24. **.controller*.controllerApps[0].typename = "LLDPForwarding"
25. #learning switch
26. **.controller*.controllerApps[0].flowModIdleTimeOut = 5
27. **.controller*.controllerApps[0].dropIfNoRouteFound = true
28. **.controller*.controllerApps[1].typename = "HF_LLDPAgent"
29. **.controller*.controllerApps[1].flowModIdleTimeOut = 140
30. **.controller*.numTcpControllerApps = 1
31. **.controller*.tcpControllerApps[0].typename = "HyperFlowAgent"
32. **.controller*.tcpControllerApps[0].connectAddressHyperFlowSynchronizer = "hf_synchronizer"
33. **.controller*.tcpControllerApps[0].checkSyncEvery=250ms
34. # NIC configuration
35. **.ppp[*].queue.frameCapacity = 10 # in routers
36. **.DC*.tcp.mss = 80

```

3.3. Simulation versus Real Communication Service Provider Network Architecture: An Overview

Testing new protocols, concepts, or configurations in real CSP production core networks is not always a viable option as this kind of action can result in an unplanned downtime ([44]). Real systems have contractual obligations that they need to meet, and planned network outages can attract severe financial penalties, especially for service providers and subsequent loss of customers that subscribe to a particular CSP network [45, 46]. This experiment deploys the LTE-like simuLTE plugin to closely test how this virtual experiment may work in a real deployment. According to [26, 44], simuLTE does not include support for control plane (CP) protocols such as radio bearer, EPS bearer, or Radio resource control protocol (RRC). However, this experiment's absence of this CP does not hinder our ability to develop a working experiment model with the user plane protocols built in the simulator. User applications that run in simuLTE and which can be found in real systems include voice over internet (VoIP), video, file transfer protocol (FTP) and Internet Television (IPTV), etc. Details on the structure, CP, User protocols, and configuration of simuLTE can be found in [44].

4. Test Environment and Simulation Exercise

4.1. Simulation Implementation and Parameters

We created LTE like environment in the OMNeT++ simulator together with OpenFlow and simuLTE add-ons. OMNeT++ is a free simulator that simulates different network types and protocols [40, 47, 48]. The simuLTE and OpenFlow plugins enable LTE and OpenFlow systems, respectively [26, 32]. The simulation experiment was set up on Ubuntu 18.04 operating system with OMNeT++ 5.5.1 and INET 3.6, simuLTE 0.9.1, and OpenFlow 1.3.4. Fig.6 shows the OMNeT IDE for orchestrating the simulation experiments.

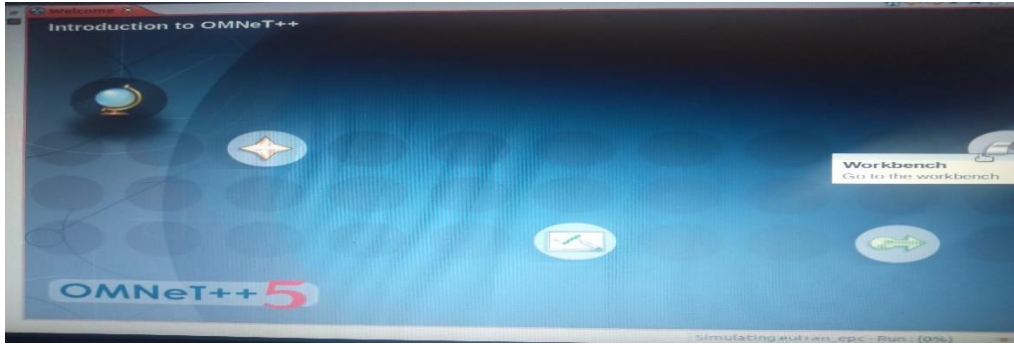


Fig.6. OMNeT++ Login screen

Several topologies exist which could be used to design and deploy modern CSP networks [2]. We have modeled the network for this experiment using two commonly used topologies; partial mesh and star topology [37]. Two test networks are created based on the classical network running OSPF in the core. It is also shown in Fig.7 and multi-controller-based network architecture installed using OpenFlow controller in Fig.8 with HyperFlow.

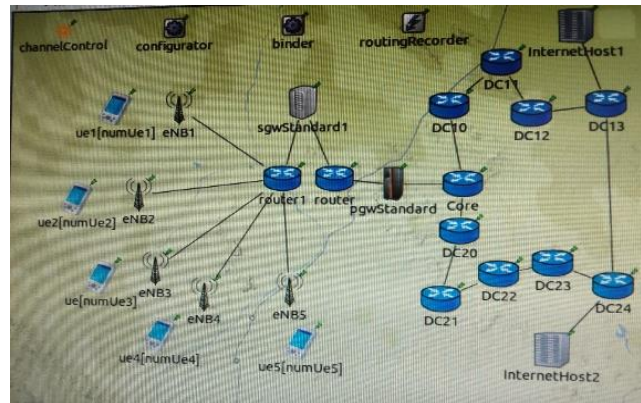


Fig.7. Partial classical network with OSPF

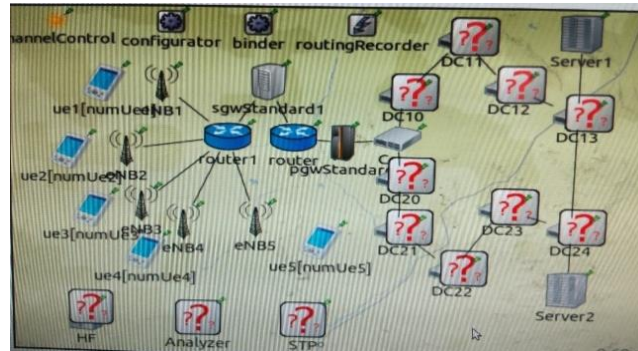


Fig.8. Partial diagram of Multi-Controller Network

In a classical network experiment, the modules deployed are; UEs, packet gateway filter (PGW), and generic servers hosted on the internet. Similarly, we have added UEs, PGW, HyperFlow, OpenFlow controller, and OpenFlow switches in a multi controller-based network simulation. In sections B to E, we highlight the critical simulation components and their configuration setup.

4.2. User Equipment

UE emulates a smartphone and supports user datagram protocol (UDP) and transport control protocol (TCP). Fig.7 illustrates UEs structure and configured user protocols.

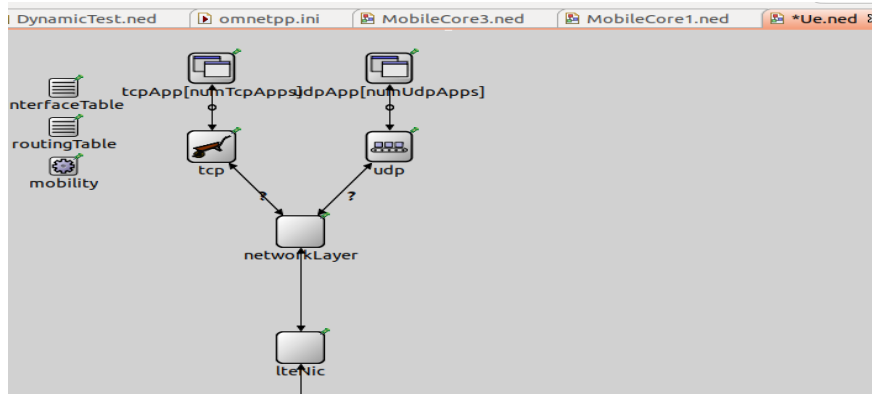


Fig.9. UE and enabled Protocols

4.3. OSPF Router

Routers in OMNeT++ framework are configured with OSPF protocol and connected at the data center. Routers build the network topology using the Dijkstra algorithm until all the routers achieve convergence. Fig.8 shows OSPF router internal structure.

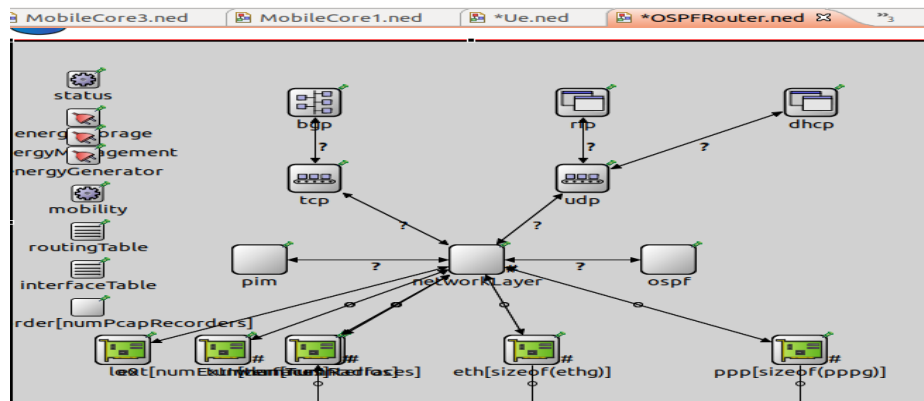


Fig.10. OSPF Router Internal Structure

4.4. OpenFlow controller

The controller is a POX-based OpenFlow protocol and readily available as an add-on in OMNeT++ [32]. Details on POX development and configuration can be found in [32]. Fig.11 describes the structure of the OpenFlow controller.

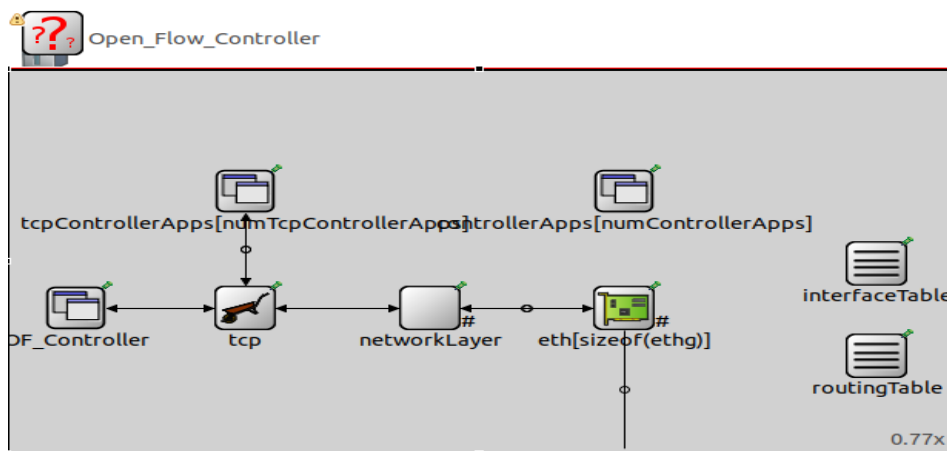
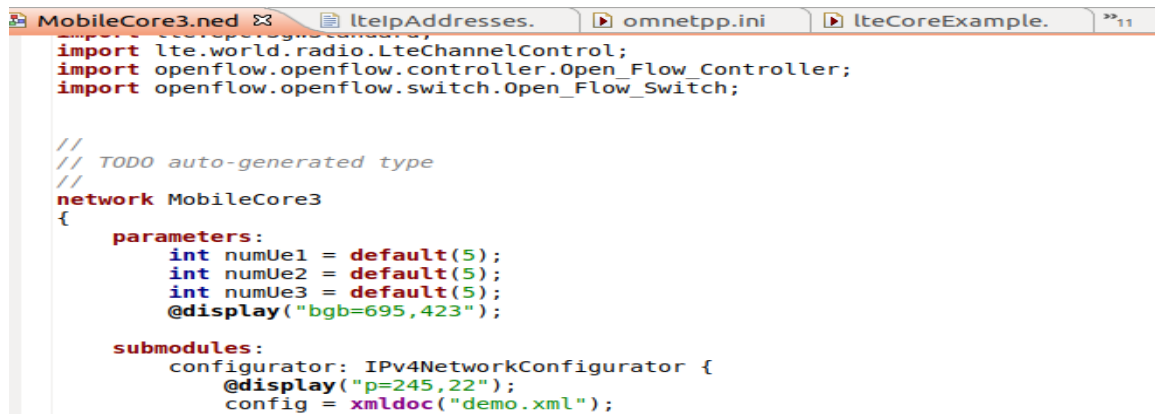


Fig.11. OpenFlow Controller

4.5. OSPF Network Operation

UEs are connected to the network via eNodeB RAN. The IP configurator shown in Fig. 12 assigns IP to UEs in DHCP-like mode and can create subnets automatically or read from an XML file script.



```

import lte.world.radio.LteChannelControl;
import openflow.openflow.controller.Open_Flow_Controller;
import openflow.openflow.switch.Open_Flow_Switch;

//
// TODO auto-generated type
//
network MobileCore3
{
  parameters:
  int numUe1 = default(5);
  int numUe2 = default(5);
  int numUe3 = default(5);
  @display("bgb=695,423");

  submodules:
  configurator: IPv4NetworkConfigurator {
    @display("p=245,22");
    config = xmldoc("demo.xml");
  }
}

```

Fig.12. IPv4 Configurator and Script File

UE sends IP packets with the destination address of server1 or server 2. The sending selection is balanced through a random sequence inbuilt in OMNeT++ [40]. The packets are distinguished using port numbers 3899 for video and 4000 for ping traffic on the server-side. When the packet arrives at the data center, the OSPF router encapsulates the data packet and sends it to the next-hop address until it reaches the destination. The packet request packet is sent back to the UE using the randomly generated port number and source IP received in the server request. After that, the results are recorded and displayed in the OMNeT++ IDE. The routers exchange information beforehand to establish neighbors and periodically send ping hello packets to confirm that a neighbor is not offline. Fig.13 illustrates UEs' process connecting on the RAN and communicating with internet servers together with the progress tab below.

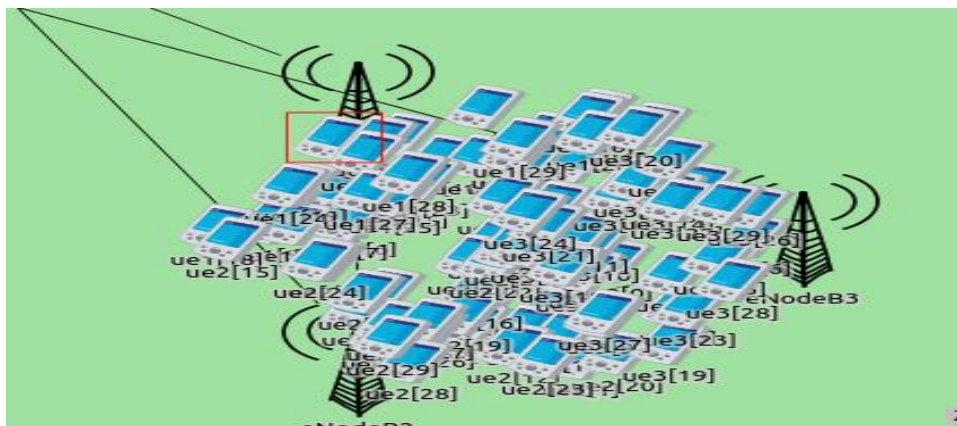


Fig.13. UE connecting to Radio Area Network

4.6. OpenFlow Network Operation

As noted earlier, OpenFlow provides a central reference point for forwarding data to the next destination and changing network route configuration. The fundamental communication process is the same in OpenFlow as OSPF except on how switches operate, explained in the next paragraph.

In OMNeT++, the OpenFlow switches have been modified to send data to the controller [32], typically of SDN networks. UE sends IP packets with the destination address of server1 or server 2. The sending selection is balanced through a random sequence inbuilt in OMNeT++ [47]. The packet is distinguished using port numbers 3899 for video and 4000 for ping traffic. When the packet arrives at the data center, the switch performs two actions; they check the packet's source and destination. Suppose the packet is on the local switch. In that case, it is forwarded to the destination, or else the switch forwards the packet to the OpenFlow controller for further routing until the packet arrives at the correct destination. The OpenFlow controller has a full domain view of the network, and once they receive a packet, they forward it to the right destination address. For this experiment, we have two domain controllers load balancing traffic using the HyperFlow protocol. HyperFlow operates by sending the first packet in the queue to controller1 and the next packet to controller2 [25]. The UE connection process to the RAN and the data center server is similar to the OSPF network except for the routers we replace with two load balancing controllers and OpenFlow switches.

Table 2 shows a summary of the testing modules and associated parameters. The numbers were based on the need to optimize the simulation framework's operation and extend previous work [22, 26, 49].

Table 2. Testing modules and associated parameters

Module	Parameters
Network Mode	LTE
Application 1	Video
Application 3	Ping
Simulation run (in minutes)	10,20,30,40,50,60
UEs per ENB	10,20,30
Routers/OpenFlow switches	10
Controller	2
HyperFlow	Enabled on Controllers

5. Simulation Findings and Discussions

5.1. Findings

Fig. 14 presents a streaming video application over a mobile IP network and recording of packet traffic. The test was conducted for both OSPF and multi-controller-based networks running OpenFlow protocol. While using OSPF, the jitter value is 31 ms, while OpenFlow registers 21 ms. The standard deviation for the two protocols while measuring Jitter was 7.0.7 ms.

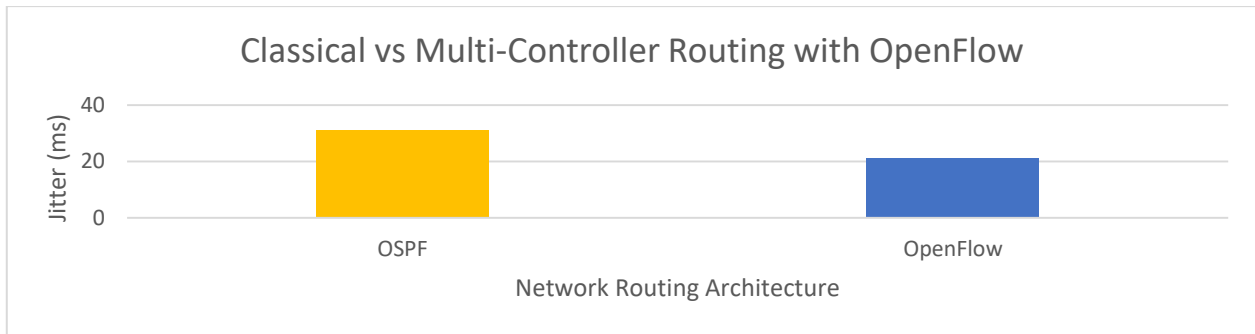


Fig.14. Classical and Multi-controller Routing with Jitter

Fig. 15 mobile UEs sent ping packets over the mobile IP network to an internet server, and results were recorded. The first simulation exercise consisted of OSPF routed network in the core. In the next set of tests, UEs send pings to the server over mobile IP network internet using a multi-controller-based network. In OSPF routed mobile, the PDR value is 91 percent mbs while using multi-controller network PDR gained was 90 percent mbs. The standard deviation between OSPF and multi-controller network with OpenFlow was 0.70 percent.

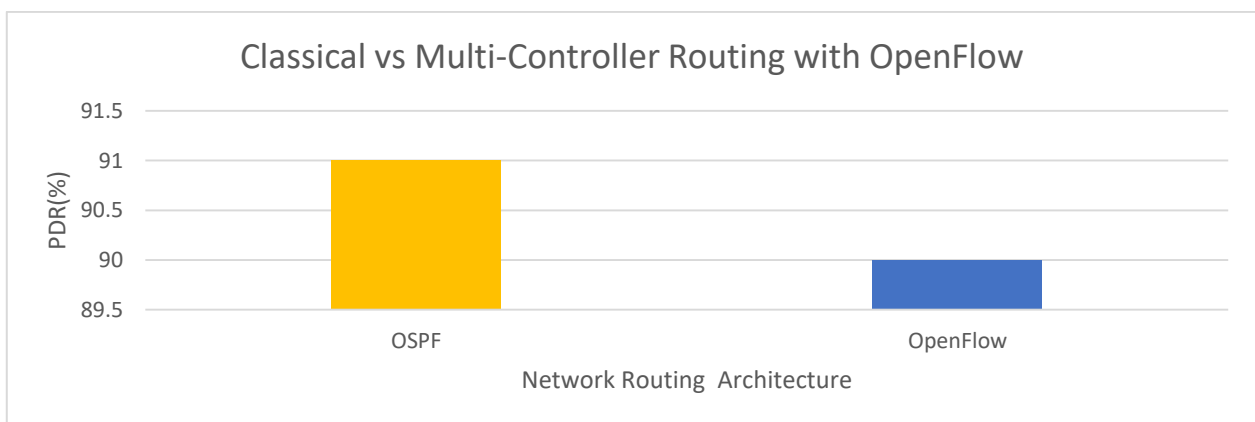


Fig.15. Classical and Multi-Controller Routing with PDR

Fig.16, we compared the number of UEs added to the network gradually using both OSPF and Multi-controller networks with OpenFlow. The Jitter values generated were 32 ms for OSPF and 22 ms for the first 10 UEs.

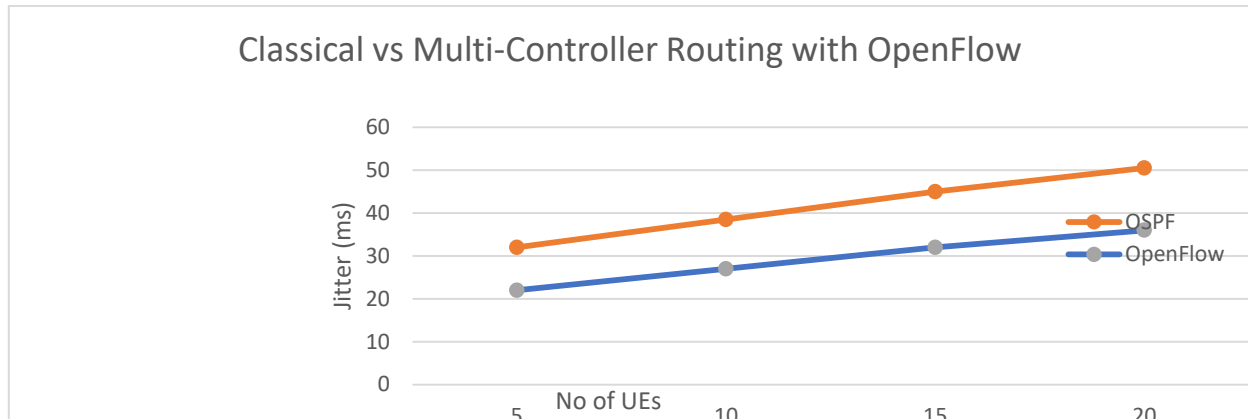


Fig.16. Number of UEs and Routing Architecture with Jitter

Fig. 17 presents PDR as more UEs are added to the network gradually. The PDR values gained 89 percent mbs for OSPF and 86 percent mbs for OpenFlow.

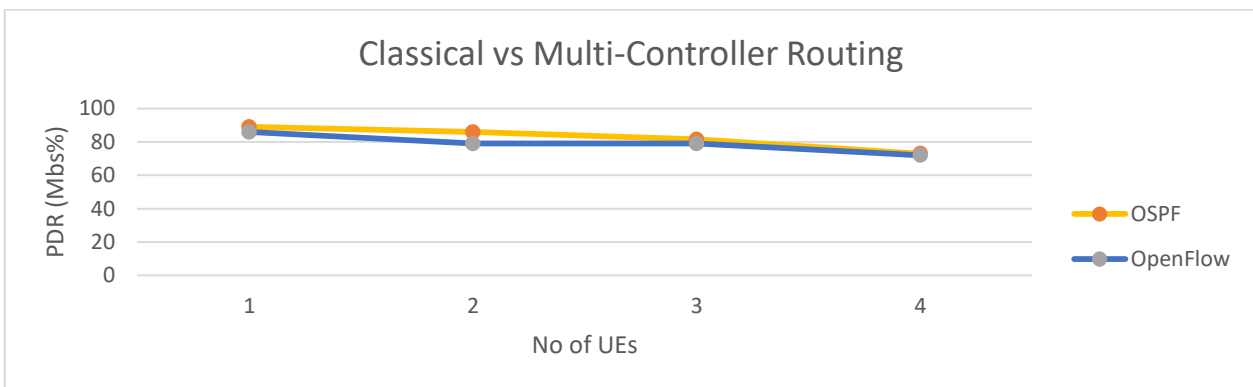


Fig.17. Number of UEs and routing architecture with PDR.

5.2. Discussion.

Multi-controller-based network architecture running OpenFlow improved Jitter by a difference of 10 ms, as seen in Fig. 14 compared with OSPF routed network. According to [15, 31, 50], controller-based networks have an entire domain view and, therefore, will quickly learn routes and or changes to the network whenever one occurs. Also, controller-based architecture can compute and identify less congested routes to reach the destination network using several parameters, of which the details are illustrated in [28, 43]. Therefore, we deduce that OpenFlow's design and nature to view the network domain improve Jitter fully. The stdev for jitter values for both protocols was 7.07, an indication that there is a significant difference between OSPF and multi-controller-based networks in handling Jitter.

On the other hand, in testing for PDR, OSPF registered a value of 91 percent compared to a controller-based network with a PDR of 90 percent in mbs. The standard deviation of 0.707 percent in mbs indicates no significant difference in PDR values gained between OSPF and controller-based mobile IP network. According to [19, 22], controller-based networks use extra overhead data when computing routes to ensure that Jitter is kept at the lowest minimum. Therefore, we again deduce that the less significant std of 0.707 percent was because OpenFlow needs to use extra mbs to remove the excessive effects that come as a result of excessive Jitter.

Effective management of user load in mobile networks is critical for how resilient and scalable mobile network infrastructure is viewed regarding routing [2]. OSPF and OpenFlow were tested by gradually adding UEs in numbers of 10 until the entire network has a total of 150 UEs to the network and observed different network traffic types, namely ping and video, as seen in Fig.17. The additional UEs added onto the network increased jitter values for both OpenFlow and OSPF at 22 ms and 32 ms, respectively. OpenFlow registered Jitter with a difference of 10 ms. That is because multiple controller-based networks have the entire domain view of the network, allowing for faster routing of data packets. Similarly, PDR was tested by adding more UEs gradually in the network. As seen earlier in Fig.17, there is a gradual decline of PDR for both protocols with values of 90 to 75 percent for OSPF and 91 to 79 percent for a multi-controller network, of which the details are further shown in Table 3.

While OSPF appears to have better PDR at 89 % mbs than a multi-controller-based network at 86 % mbs. A further look at the stdev shows a value of 0.707 mbs, indicating no significant difference between the two network architectures.

Table 3. UEs and PDR values when using OpenFlow and OSPF

No of UEs	OpenFlow PDR (%)	OSPF PDR (%)
5	90	91
10	83	87
15	81	85
20	80	84
25	75	79

6. Conclusion and Future Recommendations

The use of Multi-Controllers is becoming a new norm to provide redundancy in CSP core networks mainly. While CSP networks continue to evolve and millions of new smartphone users get subscribed on the CSP IP platforms, classical routing use cases are still evident. Single controllers have shown promise to address the challenges of classical routing in CSP networks. SDN-based controllers can improve routing efficiency by over 60% across two QoS metrics, namely PDR and Jitter. This can be attributed to dynamic controller operations such as more packet handling parameters, programmability, and centrality. However, single controllers become inefficient mainly when increased IP traffic is directed to them, which expands the routing table sizes. We introduced Multi-controller architectures and classical routing approaches in the CSP core network to evaluate the two techniques' performance differences. We simulated the CSP provider core network using OSPF and Multi-controller-based architecture on a simulator testbed. Our findings showed that Multi-controller-based core networks improved performance based on Jitter metrics by a difference of 20 ms, even as more UEs were added onto the network. In testing for PDR, OSPF registered a PDR value of 7% better than OpenFlow.

Further statistical inference on the PDR level shows that the difference between Multi-controller architecture and OSPF is stdev 0.70%. We concluded that this difference was not significant. We further deduce that Multi-controller architecture improves routing speed compared to the classical routed network by over 20% from the test results.

Controller efficiency is sometimes determined by the programming language used to develop it [51]. Therefore, future research can focus on evaluating two differently built controllers' performance, like Open Daylight, which is written using a different language from the NOX-based OpenFlow applied in this experiment [52]. We also plan to extend this work with [53] to establish if the benefits of Multi-Controllers QoS routing in the simulator can also be applied in a real communication service provider environment.

Appendix A Snapshot of Simulation Configuration on Omnet++ IDE

```

InternetHost2: StandardHost {
  @display("p=574.395:i=device/router");
}
connections allowunconnected:
router1.pppg++ <--> networkConnection <--> eNB1.ppp;

pgwStandard.pppg++ <--> networkConnection <--> router.pppg++;
router.pppg++ <--> networkConnection <--> sgwStandard1.pppg++;
router1.pppg++ <--> networkConnection <--> sgwStandard1.pppg++;
eNB5.ppp <--> networkConnection <--> router1.pppg++;
eNB3.ppp <--> networkConnection <--> router1.pppg++;
eNB4.ppp <--> networkConnection <--> router1.pppg++;
pgwStandard.filterGate <--> networkConnection <--> Core.pppg++;
DC10.ethg[1] <--> Eth10G <--> Core.ethg[0];
DC20.ethg[0] <--> Eth10G <--> Core.ethg[1];
eNB2.ppp <--> networkConnection <--> router1.pppg++;
DC10.ethg[0] <--> Eth10G <--> DC11.ethg[1];
DC11.ethg[0] <--> Eth10G <--> DC12.ethg[1];
DC12.ethg[0] <--> Eth10G <--> DC13.ethg[1];
DC20.ethg[1] <--> Eth10G <--> DC21.ethg[0];
DC21.ethg[1] <--> Eth10G <--> DC22.ethg[0];
DC22.ethg[1] <--> Eth10G <--> DC23.ethg[0];
DC23.ethg[1] <--> Eth10G <--> DC24.ethg[0];
DC13.ethg[2] <--> Eth10G <--> DC24.ethg[1];
DC13.ethg[0] <--> Eth10G <--> InternetHost1.ethg[1];
DC24.ethg[2] <--> Eth10G <--> InternetHost2.ethg[1];
}

```

Fig.18. Datacenter switch and routers connections

```

package lte.simulations.networks;

import inet.networklayer.configurator.ipv4.IPv4NetworkConfigurator;
import inet.networklayer.ipv4.RoutingTableRecorder;
import inet.node.ethernet.Eth10G;
import inet.node.inet.Router;
import inet.node.inet.StandardHost;
import inet.common.misc.ThruputMeteringChannel;
import inet.node.ospfv2.OSPFRouter;
import lte.corenetwork.binder.LteBinder;
import lte.corenetwork.nodes.Ue;
import lte.corenetwork.nodes.eNBpp;
import lte.epc.PgwStandard;
import lte.epc.SgwStandard;
import lte.world.radio.LteChannelControl;

network eutran_epcNetwork2
{
  parameters:
  int numUe1 = default(0);
  int numUe2 = default(0);
  int numUe3 = default(0);
  int numUe4 = default(0);
  int numUe5 = default(0);
  @display("i=block/network2;bgb=798,558;bgf=background/terrain");
}

```

Fig.19. Configuration and Automation of UEs

```

<?xml version="1.0"?>
<OSPFASConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="OSPF.xsd">

  <!-- Areas -->
  <Area id="0.0.0.0">
    <AddressRange address="Core>pgwStandard" mask="Core>pgwStandard/" status="Advertise" />
    <AddressRange address="Core>DC10" mask="Core>DC10/" status="Advertise" />
    <AddressRange address="Core>DC20" mask="Core>DC20/" status="Advertise" />
    <AddressRange address="DC10>DC11" mask="DC10>DC11/" status="Advertise" />
    <AddressRange address="DC12>DC13" mask="DC12>DC13/" status="Advertise" />
    <AddressRange address="DC13>InternetHost1" mask="DC13>InternetHost1/" status="Advertise" />
    <AddressRange address="DC13>DC24" mask="DC13>DC24/" status="Advertise" />
    <AddressRange address="DC24>DC23" mask="DC24>DC23/" status="Advertise" />
    <AddressRange address="DC24>InternetHost" mask="DC24>InternetHost/" status="Advertise" />
    <AddressRange address="DC23>DC22" mask="DC23>DC22/" status="Advertise" />
    <AddressRange address="DC22>DC21" mask="DC22>DC21/" status="Advertise" />
    <AddressRange address="DC21>DC20" mask="DC21>DC20/" status="Advertise" />
  </Area>

  <!-- Routers -->
  <Router name="Core" RFC1583Compatible="true">
    <BroadcastInterface ifName="eth0" areaID="0.0.0.0" interfaceOutputCost="1" routerPriority="1" />
    <BroadcastInterface ifName="eth1" areaID="0.0.0.0" interfaceOutputCost="1" routerPriority="1" />
    <BroadcastInterface ifName="eth2" areaID="0.0.0.0" interfaceOutputCost="1" routerPriority="1" />
  </Router>

  <Router name="DC10" RFC1583Compatible="true">
    <BroadcastInterface ifName="eth0" areaID="0.0.0.0" interfaceOutputCost="2" routerPriority="2" />
    <BroadcastInterface ifName="eth1" areaID="0.0.0.0" interfaceOutputCost="2" routerPriority="2" />
  </Router>

  <Router name="DC11" RFC1583Compatible="true">
    <BroadcastInterface ifName="eth0" areaID="0.0.0.0" interfaceOutputCost="3" routerPriority="3" />
    <BroadcastInterface ifName="eth1" areaID="0.0.0.0" interfaceOutputCost="3" routerPriority="3" />
  </Router>

  <Router name="DC12" RFC1583Compatible="true">
    <BroadcastInterface ifName="eth0" areaID="0.0.0.0" interfaceOutputCost="4" routerPriority="4" />
    <BroadcastInterface ifName="eth1" areaID="0.0.0.0" interfaceOutputCost="4" routerPriority="4" />
  </Router>
</OSPFASConfig>

```

Fig.20. OSPF Configuration

- xml/ios/ipapp_fhrp/configuration/xe-3s/fhp-xe-3s-book/fhp-hsrp-md5.html
- [36] Bliat, O., Ben Mamoun, M., & Benaini, R. (2016). An Overview on SDN Architectures with Multiple Controllers. *Journal of Computer Networks and Communications*, 2016. <https://doi.org/10.1155/2016/9396525>
- [37] Odom, W. (2019). *CCNA 200-301, Volume 1 Official Cert Guide*. Retrieved from www.ciscopress.com
- [38] Moy, J. (1999). *Anatomy of Routing Protocol* (First.). Massachusetts: Addison Wesley.
- [39] Moy, J. (1998). *Request for Comments 2328*.
- [40] OMNeT++. (2019). *OMNeT++ Version 5.5*. OMNeT++.
- [41] Szigeti, T., Hattingh, C., Barton, R., & Briley, K. (2013). *End-to-End QoS network design*.
- [42] Anju Bhandari, V.P. Singh, "Design of Fuzzy-Based Traffic Provisioning in Software Defined Network", *International Journal of Information Technology and Computer Science(IJTCS)*, Vol.8, No.9, pp.49-61, 2016. DOI: 10.5815/ijitcs.2016.09.07
- [43] Tourrilhes, J., Sharma, P., Banerjee, S., & Pettit, J. (2014). The Evolution of SDN and OpenFlow : A Standards Perspective. *HP*, 15.
- [44] Nardini, G., Antonio, V., Rudolf, H., Varga, A., & Meszero, L. (2021). SimuLTE - LTE System Level Simulation Model and Simulator for INET & OMNeT++. *Github*. Retrieved February 24, 2021, from <https://github.com/inet-framework/simulte>
- [45] Otiato, G. (2020). Airtel, Telkom chip at Safaricom's mobile data market - Business Daily. *Businessdaily*. Retrieved January 12, 2021, from <https://www.businessdailyafrica.com/bd/corporate/companies/airtel-telkom-chip-at-safaricom-s-mobile-data-market-3248290>
- [46] Businessdaily. (2021). CA puts Telkom, Airtel on notice over call quality - Business Daily. *BusinessDaily*. Retrieved March 1, 2021, from <https://www.businessdailyafrica.com/bd/economy/ca-puts-telkom-airtel-on-notice-over-call-quality-3275124>
- [47] Andras, V. (2017). *A Quick Overview Of The OMNeT ++ IDE*. *Omnet++*.
- [48] Vesely, V., Matousek, P., & Sveda, M. (2013). Multicast Simulation and Modeling in OMNeT++, (January 2014), 1–5. <https://doi.org/10.4108/simutools.2013.252046>
- [49] Mckeown, N., Anderson, T., Balakrishnan, H., Parulkar, G. M., Peterson, L. L., Rexford, J., ... Louis, S. (2008). OpenFlow: enabling innovation in campus networks. *Computer Communication Review*, 38, 69–74. <https://doi.org/10.1145/1355734.1355746>
- [50] Vissicchio, S., Vanbever, L., & Rexford, J. (2014). Sweet Little Lies : Fake Topologies for Flexible Routing, 1–7.
- [51] Z. Abdullah, M., A. Al-awad, N., & W. Hussein, F. (2019). Evaluating and Comparing the Performance of Using Multiple Controllers in Software Defined Networks. *International Journal of Modern Education and Computer Science*, 11(8), 27–34. <https://doi.org/10.5815/ijmeecs.2019.08.03>
- [52] OpenDaylight. (2021). Platform Overview - OpenDaylight. *OpenDaylight*. Retrieved February 25, 2021, from <https://www.opendaylight.org/what-we-do/odl-platform-overview>
- [53] Telkom. (2021). Graduate Trainee Programme at Telkom Kenya. *Telkom Kenya*. Retrieved March 1, 2021, from <https://www.telkom.co.ke/graduate-trainee-programme-telkom-kenya>

Authors' Profiles



Nicholas J. Omumbo received his Bachelor of Science in Computer Science from the Jomo Kenyatta University of Agriculture and Technology. He is currently an M.Sc. student at Maseno University, Kenya. His research interests include routing, software-defined networks, and security.



Titus M. Muhambe is a Ph.D. holder from the University of Nairobi. He is currently a member of the faculty, Information Technology Department, Maseno University, Kenya. His research interests include networks, telehealth, mobile computing, and cybersecurity. He is currently the MSc supervisor to Nicholas Omumbo.



Cyprian Ratemo is a Ph.D. holder from the University of Kibabi. He is currently the director of eLearning at Kisii University Kenya. His research interest includes mobile computing, eLearning, and software-defined networks. He is presently the MSc supervisor to Nicholas Omumbo.

How to cite this paper: Nicholas. J. Omumbo, Titus. M. Muhambe, Cyprian M. Ratemo, "Evaluation of Routing Performance using OSPF and Multi-Controller Based Network Architecture", International Journal of Computer Network and Information Security(IJCNIS), Vol.13, No.4, pp.45-61, 2021. DOI: 10.5815/ijcnis.2021.04.05