Simulation and Performance Analysis of Software-Based Mobile Core Network Architecture (SBMCNA) using OMNeT++

Mukhald Salih, Nawar Jawad & John Cosmas
School of Electrical &Computer Engineering
Brunel University, Uxbridge, London UB8 3PH
Email: mukhlad.salih@brunel.ac.uk

***Abstract*— Software defined Networking (SDN) represent the future framework for the mobile networks. This paper discusses the required modifications within the EPC in order to overcome some of the limitations of the current EPS, these modifications include introducing an SDN based solution Software Based Mobile Core Network Architecture (SBMCNA), we also show that Openflow protocol 1.3 has been extended to develop two methods to support GPRS Tunnelling Protocol (GTP) operations. The use of an intelligent Forwarding device (FD) were proposed to reduce the signalling load. SBMCNA were built on OMNeT++ by extending simuLTE [12] and Openflow 1.3 [13] modules. Load balancing and resiliency were used to demonstrate the capability of the proposed system to reduce the signalling load. The preliminary results of the system performance are presented.**

I. Introduction

In the recent years, the world has witnessed a massive growth in data traffic of mobile networks, due to increasing numbers of connected devices, such as smartphones and tablets. Customers' expectations for high speed mobile broadband are on the rise as people rely more and more on real-time mobile applications, high quality video content, cloud-based services and expect to stay connected anytime, anywhere. The expensive equipment and vendor specific configuration interfaces of the current cellular network requires a re-design of the network's data and control plane infrastructures. In fact, now it is widely accepted that the future cellular networks will require a greater degree of service awareness and optimum use of network resources.

Researchers in both academia and industry have presented several proposals to adapt SDN in the core of mobile network. For instance, the work presented by [1][2][3] introduces the concept of extending Openflow protocol to support GTP tunnel operation, including traffic matching and en- and de-capsulation operations to adapt and benefit from SDN characteristics in mobile networks. An alternative method of realizing the SDN solution is described in [4][5], which is based on the concept of the elimination of GTP tunneling and employs the global view of the controller to handle users' operations. However, these studies are limited in that they present straightforward realizations of SDN/Openflow but lack a detailed analysis of the necessary procedures.

A different approach from the aforementioned studies, which shifts focus to presenting the virtualized cellular network utilizing Network Function Virtualization (NFV) to provide high availability, elasticity, and modularity and which relatively reduces the capital expenditure, and operation expense, is subsequently offered by [7][8]. This approach also leverages SDN to manage the virtualized network. The authors show how to separate the network functions from the hardware using NFV.

This paper makes the following contributions:

- Present a new mobile core network architecture called Software based Mobile Core Network Architecture (SBMCNA), whose inherent Software Defined Networking (SDN) characteristic to provide an abstraction layer separating the control plane from the underline data plane.
- Present an OMNeT++ module to simulate the SBMCNA. The simulation platform is based on the integrations of simuLTE [12] and the extended version of the author's own works of Openflow 1.3 [13].
- Compare the performance of two methods to support GTP in Openflow 1.3 forwarding device.

The remainder of this paper is organized as follows: Section II briefly describes LTE network architecture, and the concept of SDN. Section III presents the SBMCNA and explains the functionality of its components. Section IV describes the main operations realized by the SBMCNA. Section V describes the implementation of the SBMCNA architecture and the developed solution in the OMNeT++. Section VI shows the results obtained from the simulation and compares the system performance of two distinct design methods. Finally, the paper ends up with conclusions.

II. General Background

This section gives a brief overview of the current LTE architecture and explains the main functionality of its components. It also explains the new approach to networking for Mobile Network Operators known as Software Defined Networking (SDN).

A. Evolved Packet Systems

The EPS architecture has been designed to provide seamless IP connectivity between user equipment (UE) and external Packet Data Networks (PDNs). The main functional elements and connection interfaces of EPS system architecture are illustrated in Figure 1. EPS consists of two important networks namely the access and core networks. The access network is also known as the Evolved Universal Terrestrial Radio Access Network (E-UTRAN) and it consists of multiple base stations denoted as the eNodeB. The core network also known as the Evolved Packet Core (EPC) consists of multiple functional elements that includes the Mobility Management Entity (MME), Serving Gateway (SGW), PDN Gateway

(PGW), Home Subscriber Server (HSS), and Policy and Charging Rule Function (PCRF).

The eNodeB represents the essence of the access network and it is responsible for the air interface toward the User Equipment (UE) and S1 interface toward the core network. While the EPC connects the mobile network to external Packet Data Networks (PDN) that includes the Internet, IP Multimedia Core Network Subsystem (IMS) and private corporate networks which provide the UEs with a variety of services. The SGi interface is utilized to maintain the connection with the outside networks. To provide data services to the UE, EPS employs the concept of bearers to route IP traffic over the EPC. A bearer is a logical connection, which represents a set of network configurations that uniquely identifies traffic flows between a UE and PGW associated with a common Quality of Service (QoS). An EPS bearer crosses multiple interfaces, such as radio interface between the UE and eNodeB, S1-U interface between the eNodeB and SGW, and S5/S8 interface between the SGW and PGW. The GPRS Tunneling Protocol (GTP) is used to encapsulate user data over the S1-U and S5/S8 interfaces. IP addresses, UDP port number and Tunneling End ID (TEID) identify the GTP tunnel.

B. Software Defined Networking (SDN)

SDN is the emerging architecture that continues to be the dominant topic of interest in networking today. It is dynamic, manageable, adaptable, and cost-effective architecture. SDN represents the latest incarnation of networking technologies that is basically proposed as an approach to provide physical separation between the control plane and the data plane to realize several distinct advantages, such as efficient resource utilization, centralized control and reduced network operations complexity. The control plane is run as logically centralized software programs that controls the entire network which increase the optimization of the network resources by automatically and dynamically programing the underlay network switches. Moreover, SDN represents a mechanism to boost the flexibility achieved by network operators by allowing them to have privilege control of network functions independent of the proprietary hardware substrate. OpenFlow protocol represents the first and the most commonly used standard protocol for the communications between the control and data plane layers of the SDN architecture. This paper is based on extended version of Openflow 1.3 specification of the SDN network.

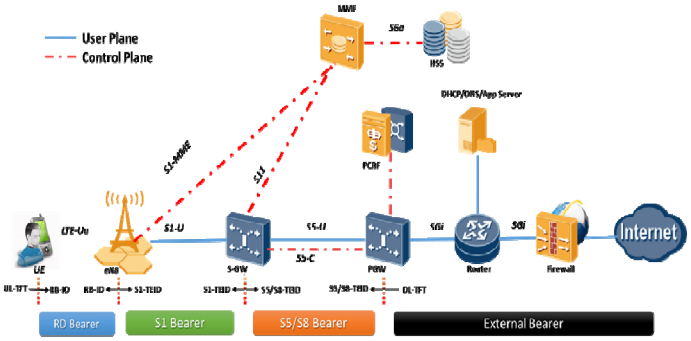

Figure 1 – Evolved Packet System (EPS) Architecture

III. SBMCN ARCHITECTURE

Figure 2 illustrates the main components of the SBMCN architecture, which is composed of the elements outlined in the below subsections.

A. Network Controller (NC)

The Network Controller (NC) plays a crucial role in SBMCN architecture. It consists of three key components, specifically an application layer, a network operating system, and communication interfaces. The application layer composes of sets of applications that implement network functions and services such as MME and SPGW-C applications that will be described later in this section. The network operating system consists of several building blocks that are responsible for topology auto-discovery, topological resource view and network resource monitoring. The communication interfaces are the northbound, southbound, and horizontal interfaces (east-west). The northbound interface is used for communication between the application layer and the network operating system such as for providing a virtual view of the network to the application layer; it also receives policies for the application layer. The southbound interface handles the communication between the data and control planes. The horizontal interface allows communication between multiple controllers and supports third party interactions. The controller manages the data plane forwarding of eNodeB, SGW (denoted as SGW-D) and PGW (denoted as PGW-D). The NC is responsible for user session establishment and load monitoring at the data plane.

B. MME

The MME keeps the same functionality specified by 3GPP with one exception, which is the SGW and PGW selection, because this task is handled by the NC. The MME communicates with the NC using the Application Programming Interface (API). The 3GPP interface between the MME and HSS is maintained [9].

C. Serving Gateway and PDN Gateway control plane (SGW-C and PGW-C)

The SGW-C and PGW-C represent the brain and control intelligence of the SGW and PGW respectively. These functions, together with the MME, are virtualized and packaged as applications on top of the NC, as shown in Figure 2. They play a major role in GTP tunnel establishment including tunnel endpoint identifier (TEID) allocation. They allocate unique TEID values for the S1-U and S5-U interfaces for both uplink and downlink traffic. PGW-C is also responsible for UE IP address allocation [9].

D. Forwarding Device (FD)

The FD represents a set of Openflow switches enhanced with important features to optimize them for use in a mobile network. These features include GTP encapsulation/ decapsulation, and a local program. Therefore, it can apply rules received from the controller in two distinctive ways that will be explained in section V.B.

E. OpenFlow-Based eNodeB

This element represents the point of interaction between the access and core network in the SBMCN architecture. It still keeps the same radio functions as specified in 3GPP standards, while its S1 interface is replaced by an OpenFlow switch. Therefore, it is necessary to define a new set of control signaling to be sent over the OpenFlow link between eNodeB and the NC to handle UE operations, while the data plane is programmed according to instructions received from the NC. Each instruction has a hard time-out that represents the flow entry lifetime. The eNodeB maintains the radio and S1 bearers for as long as the session is active, or the hard time-out has not expired. Upon detecting UE inactivity, the radio bearer is released and the S1 bearer is maintained (i.e. the OF entry is maintained) as long as the Release Timer has not expired. The Release timer is configured according to the traffic pattern (e.g. session type, session duration, periodic connection request, etc.).

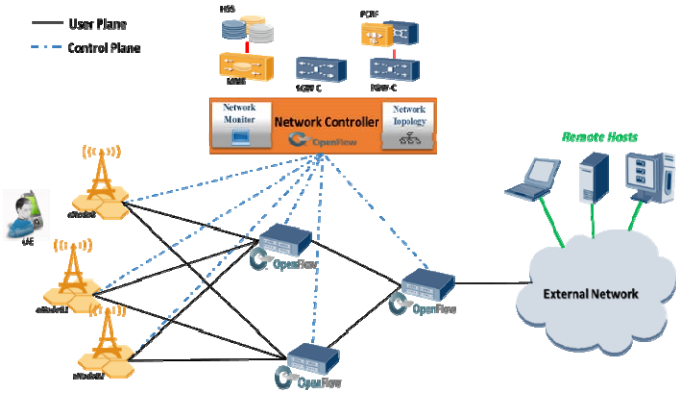

Figure 2 Software-Based Mobile Core Network Architecture (SBMCNA)

IV. ARCHITECTURE OPERATIONS

This section demonstrates and describes three procedures. Basically, these procedures include service request, resiliency and load balancing although the same concept is applicable to the other scenarios proposed in [10][11].

A. Service Request Procedure

This procedure represents a step-by-step sequence of activities triggered by a UE in an idle state that wants to use a service from the Internet or the PDN networks.

Upon receiving the service request message, the NC controller initiates the bearer setup procedure to configure the data plane forwarding devices. In our architecture two methods are implemented and compared. In the first method, the controller converts the information provided by the network function applications to a sequence of Flow-mod-Msg and sends them to each forwarding device in the selected path (a sequence of switches from one edge switch to another) to install a flow entry for each direction, as proposed by [1]. Secondly, a local program is added to each forwarding device to interpret the User-DataPlane-Setup-Request message received from the controller and convert its information to two flow entries: one to handle the uplink traffic and another to handle the downlink traffic.

B. Resiliency

In this work resiliency is described as the method of handling SGW failure without the need to release all the active sessions handled by the failed SGW and wait for the effected UE to initiate the service request procedure used by the 3GPP. When a SGW-D failure is discovered by the NC, the SGW-C is notified. The latter selects another SGW-D and provides the NC with the updated information of the new link for the impacted UEs. The SGW-C may select one or more SGW-Ds based on the last logged load of the failed SGW-D; if there is a SDW-D that can handle the entire load of the impacted UEs then a single SGW-D is selected, otherwise multiple SGW-Ds are selected. Based on the information received from the SGW-C, the NC configures the flow table of the newly selected SGW-Ds and modifies the virtual port configuration of the PGW-D to reroute the downlink traffic to the new SGW-D. This process is carried out by changing only the destination IP address and the physical port number if required, since the SGW-TEID values for the downlink traffic on the S5 interface remain the same for the impacted sessions. The NC will repeat the same operation with the serving eNodeBs to redirect the uplink traffic to the desired SGW-D in the same fashion, as the SGW-C does not create new TEID values during the restoration procedure.

C. Load-Balancing

Load-balancing is a widely adopted workload distribution method used to effectively optimize the network resource usage. In the present implementation, the concept of load awareness offered by the SDN controller is utilized to implement the load-balancing mechanism. The monitoring application properly and efficiently collects fine-grained statistics from the network devices. The monitoring application has a process that is responsible for building a historical profile for each user. This profile includes the current usage, user's active hours, and mobility patterns, among others. The load-balancing application utilized both the statistics obtained by the monitoring application and the weighting factor to perform load distribution. If one of the SGW-Ds suffers from congestion, the controller seamlessly moves some of the UEs to another SGW-D to provide better resource distribution without increasing the signaling load.

V. NETWORK MODULE

The network topology used in our simulation is shown in Figure 3. It consists of the core network (EPC), access network (EUTRAN) and the InternetHost. The core network comprises of an OpenFlow controller and three enhanced Openflow forwarding devices that are capable of GTP tunneling. The core network adds realistic delays that simulate the time required for a packet to travel from the InternetHost to the access network and vice versa. Specifically, 8ms for the S1-U interface, 4ms for the S5 interface and 8ms for the SGi interface. The EUTRAN consists of three Openflow capable eNodeBs and a set of UEs, where each eNodeB is equipped with an omnidirectional antenna with 40 dBm transmission power, 18 dB antenna gain, 5 dB noise figure and 2 dB cable loss. The

RLC layer at the eNodeB is configured with the Unacknowledged Mode, with a fixed PDU size of 40 bytes. A realistic channel model that considers both path loss and fading is used in our simulation. The Urban Macro Path Loss Model and the Jakes model for Rayleigh fading is used in all scenarios. The UEs are configured to use 26dBm transmission power and a linear mobility model with a speed of 1m/s.

The network is implemented in OMNeT++ version 4.2.2 utilizing its independently developed open source module INET 2.3 together with simuLTE and OpenFlow 1.3 extensions. OMNeT++ is run on the Windows 7 Pro operating system hosted by a Dell Precision Tower equipped with Intel® Xeon® E3-1246V3 / 3.5 GHz CPU with 8 GB of RAM. INET library and the aforementioned extensions provide flexible tools that are utilized in the creation of the simulation platform used to evaluate and quantify the efficiency and the performance of the proposed architecture. To evaluate the integration of SDN in EPC, a prototype implementation was developed that introduced several modifications, improvements and extensions to the base version of both simuLTE and Openflow 1.3 models, in order to create a simulation platform that offers complete tools to enable the validation of the system performance under different circumstances. The extensions include the elements detailed below.

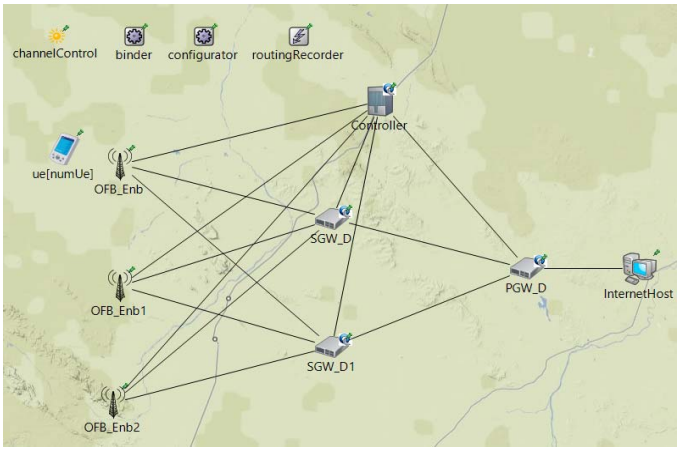

Figure 3 - Network topology used in OMNeT++ simulation

A. NC Controller Module

Figure 4 illustrates the SDN controller model in OMNeT++. It consists of an application layer, a network operation system and communication interfaces. The mmeApp module works together with the spgwApp modules to realize the network functions and services by sending instructions or requirements to the ControllerOS module to relay them to the networking components. The ControllerOS module is also responsible for extracting information from the network devices and communicating them back to the application layer. To achieve this, the ControllerOS module utilizes two framework applications, namely: topology discovery and monitoring applications, to extract and maintain network information. This information includes an abstract view of the network, statistics, and events that describe what is happening. Since HSS is not implemented in this model, the userTable module is used to store the UEs subscription profile and, for simplicity, the UE authentication procedure is not implemented. The SDN controller model implements most of the communication interfaces, although not all of them have the accuracy that is specified by the standard. The OMNeT++ signal mechanism is used as a synthetic representation of the northbound interface between the ControllerOS and the application layer. The southbound interface has been modeled with high accuracy and real packets are built and exchanged between the controller and the forwarding devices. The SDN controller model has a TCP connection with every forwarding device in the network exclusively for the purpose of exchanging control and statistic messages. Openflow protocol is used to realize the southbound interface by handling the communications between the individual network devices and the controller.

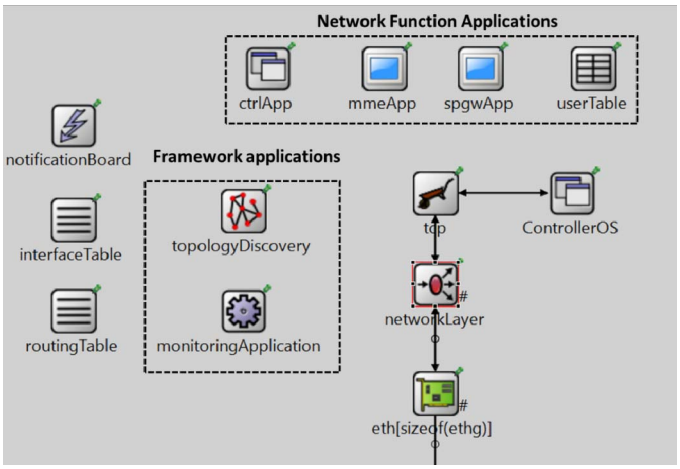

Figure 4 - Controller Module in OMNeT++

B. GTP-Capable-Openflow-Switch Module

The module depicted in Figure 5 represents an overview of the different components of the GTP-capable Openflow switch. It mainly consists of the switch control and data processing logics. Tunnelling in this implementation is realized through the deCap-vPort and enCap-vPort modules that simplifies headers manipulation to support GTP encapsulation and decapsulation.

1) Switch Control Logic

The switchApp module depicted Figure 5 represents an extended version of the original OpenFlow control logic module that we presented in [13].

It has a local program with a focus on the interpretation of the new Openflow messages and converts them to flow entry. It is also responsible for providing the controller with aggregate statistics of the forwarding device. Therefore, switchApp can handle both the standard OpenFlow control messages and the User-DataPlane-Setup-Request. The latter is a new message that we propose to carry information to setup UE uplink and downlink tunnels. For example, in the service-request procedure, the controller configures the GTP tunnel by sending User-DataPlane-Setup-Request to the selected forwarding devices. The message sent to the SGW-D contains the SGW-S1-TEID, SGW-S5-TEID, eNodeB-S1-TEID, eNodeB-IP-Address, physical, PGW-S5-TEID, PGW-IP-Address, physical output ports, and QoS parameters. Upon the

reception of this message the local program configures the TEID Table to add two entries that hold the virtual port encapsulation parameters. The Tunnel-Id uses a key in the TEID table. Let us say that the TEID table has two entries, 100 and 200: the former to encapsulate the packet with eNodeB information (TEID, IP-Address, UDP Port Number (always 2152 in our simulation)) and the latter to encapsulate the packet with PGW information. Then the local program installs two flow entries to map the traffic to the correct virtual port. The first entry matches traffic with SGW-S1-TEID to be sent to the virtual port 200 and the second entry matches traffic with SGW-S5-TEID to be sent to the virtual port 100.

2) Switch Data Processing Logic

The data message received by the physical port is passed up without any modifications to the processingUnit module, which implements the OpenFlow switch functionality on the data plane. The processingUnit is pre-configured with flow entry that has the highest possible priority. This entry matched traffic is destined to UDP Port 2152 (GTP-U traffic) and sends it to the deCap-vPort. The returned packet from the deCap-vPort consists of the IP header with a transport-layer header plus payload; the GTP TEID is also written into the lower 32 bits of the tunnel-Id metadata. Another flow entry matches the GTP-U tunnel utilizing the tunnel-Id metadata and sends it to the enCap-vPort.

Table 1 - Main Components of TEID Entry

Tunnel-Id	Next Hop TEID	Next Hop Address	Next Hop Port Number	QoS Parameter	Physical port Number

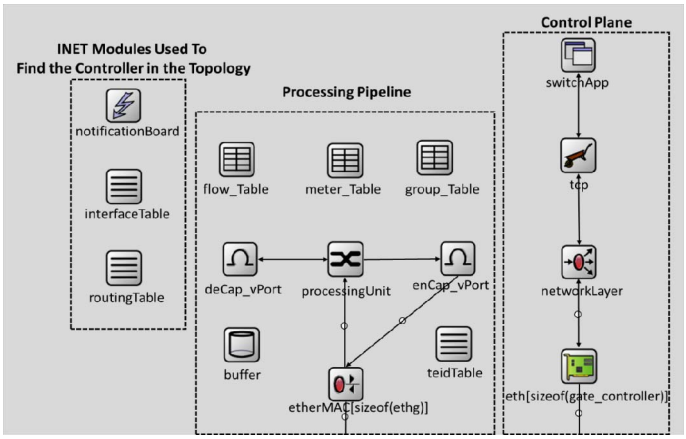

Figure 5 - Implemented Model of a GTP capable Openflow Switch

The enCap-vPort operation is directly mapped to the information specified by the teidTable. When the packet arrives at the enCap-vPort, it first extracts the output tunnel-Id from the control information attached to the packet, and then it looks up the tunnel header information in the teidTable. If a match is found, then the packet is encapsulated with GTP utilizing the tunnel information of the teidTable as shown in Table 1. If no such tunnel information is present, the enCap-vPort checks if the value of output tunnel-Id if it is equal to one of the physical ports; then the packet is encapsulated within Ethernet headers utilizing the information included in the control information attached to the received packet. Otherwise, the packet is forwarded to the controller with an error indication.

The teidTable consists of one or more entries. Each entry is composed of tunnel Id, TEID value, next hop address, next hop port number, and QoS parameter, as shown in Table 1.

C. Openflow-Based eNodeB (OFB_Enb)

The Openflow-Based-eNodeB keeps the same radio protocol stack as specified by 3GPP standards leveraging the LTE NIC module from simuLTE [12], while its S1 interface is replaced by an OpenFlow switch. To handle UE authentication, authorization, and mobility management, a new set of signalling messages are added to the Openflow protocol to carry and exchange the necessary information between the network entities during the UE operations. The main control messages included in the OMNeT++ model are Initial-UE-Message, Initial-Context-Setup-Request, Initial-Context-Setup-Response, Switch-Path-Request, and Switch-Path-Request-ACK. The new messages have similar information to those specified by the 3GPP standards but are structured using OpenFlow principles.

VI. Experimental Setup and Results

Two experiments are conducted to evaluate the system performance. Each experimental simulation lasted for 100s and was replicated five times with different seeds to exclude simulation artefacts. At the start of each experiment, the UEs were randomly placed within a square of a given size at a maximum distance of 100-meter from the eNodeB and moved linearly at the speed of 1m/s.

In first experiment, the control-signalling messages sent by the controller to data plane tunnels management during the service request and SGW failover procedures is used as a metric to evaluate the system performance. The experiments validate the performance of two GTP implementation methods. In the first method, which is proposed by [1], the controller sends a sequence of Flow-Mod and GTP-Tunnel-Modify-Request messages to configure data plane devices according to users' policies and profiles. In the second method, only a User-DataPlane-Setup-Request (message includes information about both the uplink and downlink tunnels) is sent from the controller. The forwarding devices can read and translate the information to Openflow 1.3 flow entries and specify the virtual port parameters in the TEID Table. In this experiment the UEs used PING application to send a small message (40B every 1s) to the InternetHost. Figure 6 reports the signalling loads to setup GTP tunnels with and without SGW Failover.

Results illustrated in Figure 6 show that the signalling load required to handle service request and SGW failover procedures. It can be observed that the signalling loads of method 1 are higher than method 2. It makes perfect sense because in method 2 the controller needs to send fewer messages to manage the data plane forwarding device. Moreover, in method 1, the control signalling required to update the tunnel is less than what is required to setup the tunnel.

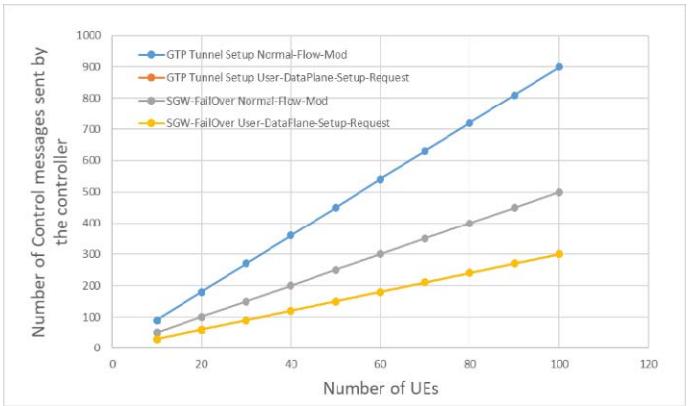

Figure 6 - Number of Control Messages Sent by the Controller vs. Number of UEs Running a Single Application that Sends PING Messages to the InternetHost

In the second experiment, weighting factor and network loads are used as mechanisms to select the SGW-D that should handle the UEs traffic. Figure 7 shows a close approximation of how the system handles load distribution with the aforementioned selection algorithms. The first insight that one can extract from Figure 7 is that the load handled by SGW-D1 is much greater than the load handled by SGW-D when weighting factor is used as a selection mechanism. In fact, this result is logical considering that a 3:1 metric is used in the selection algorithm. Changing the weighting metrics, to 2:1 or even 1:1, leads to a better load distribution. It is important to notice that a weighting-based algorithm follows a configured pattern without any knowledge about the network status, which may lead to unfair distribution of network loads, as shown in Figure 7. Conversely, load-base selection will always provide a fair distribution that aims to maximize resource utilization.

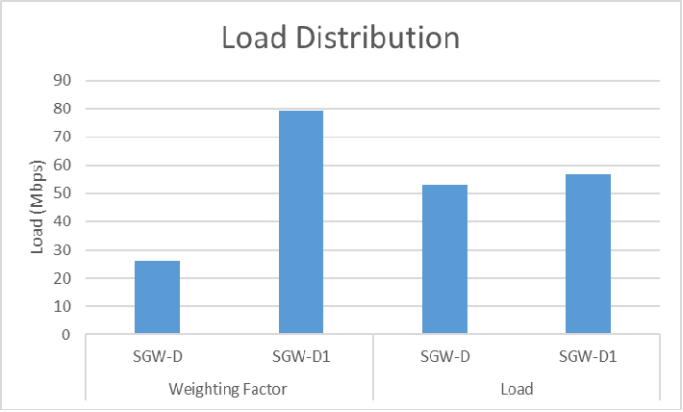

Figure 7- Load Distribution between the SGW-Ds in the Local Pool for both Weighting Factor and Network Load Selection Mechanism

VII. Conclusions

This paper has presented Software-Based Mobile Core Network Architecture. Specifically, the architecture has an inherent SDN characteristic to provide an abstraction layer separating the control plane from the underlying data plane. Three procedures are described and analyzed in detail, including a UE-triggered service request, network resiliency, and load-balancing. In each procedure two distinctive GTP tunnel implementation methods have been analyzed and compared. The results show that signalling loads are significantly reduced when the forwarding device is equipped with a local program. The experiments also show that, utilizing SDN introduces flexible and programmable aspects to the mobile core network to provide better load distribution and faster recovery time during network equipment failure

References

[1] J. Kempf, B. Johansson, S. Pettersson, H. Lüning, and T. Nilsson, “Moving the mobile Evolved Packet Core to the cloud,” in 2012 IEEE 8th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), 2012, pp. 784–791.

[2] G. Hampel, M. Steiner, and T. Bu, “Applying Software-Defined Networking to the telecom domain,” in 2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), 2013, pp. 133–138.

[3] K. Pentikousis, Y. Wang, and W. Hu, “Mobileflow: Toward software-defined mobile networks,” IEEE Commun. Mag., vol. 51, no. 7, pp. 44–53, 2013.

[4] J. Costa-Requena, V. F. Guasch, and J. L. Santos, “Software Defined Networks based 5G Backhaul Architecture,” in Proceedings of the 9th International Conference on Ubiquitous Information Management and Communication (IMCOM ’15), 2015, pp. 1–5.

[5] S. Shanmugalingam and P. Bertin, “Programmable Mobile Core Network,” in 2014 IEEE Symposium on Computers and Communications (ISCC), 2014, pp. 1–7.

[6] L. J. Chaves, V. M. Eichemberger, I. C. Garcia, and E. R. M. Madeira, “Integrating OpenFlow to LTE: Some issues toward software-defined mobile networks,” in 2015 7th International Conference on New Technologies, Mobility & Security - Proceedings of NTMS 2015 Conference and Workshops, 2015, pp. 1–5.

[7] M. R. Sama, L. M. Contreras, J. Kaippallimalil, I. Akiyoshi, H. Qian, and H. Ni, “Software-defined control of the virtualized mobile packet core,” IIEEE Comm. Mag., vol. 53, no. 2, pp. 107–115, 2015.

[8] A. Bradai, K. Singh, T. Ahmed, and T. Rasheed, “Celluar Software Defined Networking: A Framework,” IEEE Commun. Mag., no. June, pp. 36–43, 2015.

[9] S. B. H. Said et al., “New control plane in 3GPP LTE/EPC architecture for on-demand connectivity service,” in 2013 IEEE 2nd International Conference on Cloud Networking (CloudNet), 2013, pp. 205–209.

[10] M. R. Sama, S. B. H. Said, K. Guillouard, and L. Suciu, “Enabling Network Programmability in LTE/EPC Architecture Using OpenFlow,” in 2014 12th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt), 2014, pp. 389–396.

[11] V. Nguyen and Y. Kim, “Proposal and evaluation of SDN□based mobile packet core networks,” EURASIP J. Wirel. Commun. Netw., vol. 2015, no. 1, p. 172, 2015.

[12] A. Virdis, G. Stea, and G. Nardini, “Simulating LTE / LTE-Advanced networks with SimuLTE,” Simul. Model. Methodol. Technol. Appl., vol. 402 of the, no. January, pp. 83–105, 2016.

M. A. Salih, J. Cosmas, and Y. Zhang, “OpenFlow 1.3 Extension for [13] OMNeT ++,” in 015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing, 2015, pp. 1632–1637.