Hindawi
Journal of Computer Networks and Communications
Volume 2018, Article ID 1543531, 14 pages
https://doi.org/10.1155/2018/1543531

Research Article

Modeling Handover Signaling Messages in OpenFlow-Based Mobile Software-Defined Networks

Modhawi Alotaibi,[1,2] **Ahmed Helmy**,[1] **and Amiya Nayak**[1]

[1]*School of Electrical Engineering and Computer Science, University of Ottawa, Ottawa, Canada*
[2]*College of Computer Science and Engineering, Taibah University, Medina, Saudi Arabia*

Correspondence should be addressed to Modhawi Alotaibi; malot075@uottawa.ca

Received 31 July 2018; Revised 12 October 2018; Accepted 25 November 2018; Published 18 December 2018

Guest Editor: Ting Wang

Copyright © 2018 Modhawi Alotaibi et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

The software-defined networking (SDN) paradigm has become essential in tackling several issues and challenges in conventional networking, especially in mobile/cellular networks. In order to realize the benefits brought by SDN to mobility management, we study the effects of SDN in conjunction with OpenFlow protocol on the handover procedure. However, in this new setting, the handover still suffers from delay due to the exchange of OpenFlow signaling messages. In this paper, we focus on SDN in mobile networks and quantify the delays of handover-related OpenFlow messages in order to identify the performance measures as well as the underlying challenges. For our analysis, we provide an analytical model, using which we modeled two handover-related OpenFlow messages in such networks. To the best of our knowledge, no previous work has modeled OpenFlow messages other than *Packet-in* messages. In this paper, in addition to the Packet-in message, we model *Port-status* messages. Following our analysis, we propose a novel solution to make handover more efficient and less interruptive. Furthermore, we study our solution in an LTE architecture and compare it to an existing solution. We show that, in normal traffic conditions, our solution can decrease the handover delay as much as 20%.

1. Introduction

Current computer networks are built out of an enormous number of forwarding devices and middleboxes scattered on large-scales. Each device has some logic and local intelligence to perform certain roles. This networks' architecture has reportedly been suffering from multiple issues that get exaggerated with the increase in traffic demands and with new applications and protocols requirements. For instance, inflexible adaptability to network changing conditions is a major challenge that burdens network managers; additionally, the high cost due to hardware commodity is another issue [1, 2]. Moreover, scaling and managing due to the need to configure networks' devices in a low-level manner are two unavoidable challenges. On a global scale, another challenge is "Internet ossification," which means the difficulty of Internet evolution in terms of physical infrastructure and protocol installation [3].

The software-defined networking (SDN) paradigm has been introduced to solve some of the issues mentioned above and offers solutions such as reducing upgrade costs and using network resources efficiently [2]. The paradigm divides the network architecture into two planes: control and data, which makes it possible to program and configure a controller as well as manage the network dynamically and globally [4]. In fact, SDN facilitates the management of large-scale congested networks such as datacenters and cloud infrastructures. For instance, on Aug 21, 2017, Cisco announced their plan to integrate their software-defined networking product, called Application Centric Infrastructure (ACI), into the top three major public cloud platforms in the market: Amazon Web Services, Microsoft Azure, and Google Cloud Platform [5]. This incentive shows the expected wide spread of SDN concept in the future.

The SDN concept has been integrated into both wired and wireless networks. Even though the SDN technology is

introduced to solve some issues in the current wireless architecture, the nature of the wireless networks nowadays imposes critical challenges due to the rapid and extensive growth of mobile traffic [6]. Meeting the users' requirements and providing a high quality of service become necessary and more challenging [7]. One of the main concerns of mobility management is maintaining a user ongoing session continuity with relatively minimum interruption or, in other words, minimizing handover latency. Therefore, in our research, we study mobile/cellular networks, and we focus on the handover of hosts between different switches. Specifically, our focus is on minimizing handover latency by targeting one of its causes, the delay caused by exchanging control messages. The completion of a handover procedure requires the exchange of signaling messages between the control and data planes, especially in the case of hard handover, where the "breakage" in an ongoing session correlates to the exchange of management and reconfiguration messages [8]. Interestingly, Duan et al. [9] showed that an SDN-based solution is more suitable for "less sensitive" to latency applications. However, we know for a fact that the handover is sensitive to latency, and the primary goal here is to minimize handover latency. This fact motivated us to further analyze the main contributors to delay.

In cellular networks such as Long Term Evolution (LTE), the SDN concept has been incorporated into different parts, especially into the Evolved Packet Core (EPC). EPC is made of multiple components such as Mobility Management Entity (MME), Home Subscriber Server (HSS), Serving Gateway (S-GW), and Packet Data Network Gateway (P-GW). MME is a key module responsible for some mobility management tasks. Module HSS is considered a central database that has all the subscribers' information. S-GW and P-GW are gateways; S-GW forwards data between eNodeBs, while P-GW connects the end-user to external networks. In SDN-based LTE architecture, the deployment of the control plane can be centralized or distributed. As for the centralized approach, all EPC's components are virtualized and placed into a single controller. Alternatively, the distributed EPC controller architecture can be represented in different ways: either through implementing a distributed set of EPC controllers or by distributing the functionalities of an EPC controller. Thus, the mobility management can be centralized or distributed based on the choice of the control plane deployment.

According to the design of SDN, the communication between the control and data planes is going through what is referred to as a southbound interface. OpenFlow protocol is the most well-known and is also the industry-standard southbound protocol in SDN-based networks. It was proposed by McKeown et al. [2] and is now maintained by Open Networking Foundation (ONF). The OpenFlow protocol manages the interaction between the data and control planes through the use of particular messages. Analyzing those OpenFlow messages helps understanding and identifying latency issues.

In recent years, SDN has gained massive attention in both academia and the industry. This emerging technology has been studied extensively in the literature [3, 10–12]. However, most of the work related to SDN that we have come across was based solely on experiments and simulations. Recently, theoretical analysis has been gaining attention [13–15]. A mathematical model can provide researchers with insights into how an OpenFlow architecture performs according to certain given parameters under given circumstances, thus leading to propose efficient algorithms. Therefore, we provide an analytical model based on results from the queueing theory of an OpenFlow-based mobile network. We restrict our analysis of the handover delay to target the delay caused by exchanging handover-related OpenFlow messages. We are, therefore, interested in minimizing the delay caused by the exchange of signaling messages between the control and data planes. To the best of our knowledge, no other work has modeled OpenFlow messages in relation to the handover procedure completion. In fact, our analysis provides a novel mathematical modeling for such messages. Consequently, we discuss two different mobility management approaches and their impacts on handover delay. Our model and results are verified through simulations, and we place emphasis on LTE as an example of cellular networks and show results based on experimentation in Section 6.

In general, SDN is assumed to be adopted in the backhaul/transport portion rather than the operation of a mobile network. In this paper, we try to link different aspects including OpenFlow-based SDN, queueing theory, cellular networks, and mobility management approaches. By linking theses aspects, we step forward to realize issues in the handover procedure in SDN-based mobile networks in general and LTE in particular. Hence, we discuss and compare different solutions.

This paper is structured as follows. Section 2 explores some of the related work. Section 3 describes our network model, including assumptions and limitations. We then go through our analytical model in Section 4. After that, we present our proposal in detail in Section 5. We then discuss our simulation results in Section 6. We finally conclude the paper with Section 7.

2. Related Work

In this section, we go through some papers that studied SDN-based solutions. We divide the analysis here into two parts: reviewing the papers that provided analytical models based on the queueing theory and the papers that provided mobility management solutions for cellular networks such as LTE.

2.1. Modeling OpenFlow Messages. The authors of [7] studied the causes for overall latency in SDN-based mobile core networks. They linked latency to two major factors: transmission delay and processing delay. Transmission delay is described as the time taken for the data to be transmitted between the controller and the switches; placement algorithms can play an important role here. Processing delay describes the delay within the controller or a switch. Marqueza et al. observed that the number of hops influences the

increase in processing latency but that the real cause is buffering handling. They argued that the load of the connection requests plays a more significant role than the number of hops that need to be configured for each request. Regarding the processing delay, the authors suggested two strategies:

(1) To optimize the handling procedures of exchange of signaling messages in the OpenFlow channels between the controllers and switches

(2) To define forwarding mechanisms, which reduce the number of switches involved in configurations, in the data plane

In light of their results, we got motivated to investigate the theoretical aspects of the processing delay, which led us to develop an analytical model based on the queueing theory.

The author in [16] explored the challenges and benefits of using the queueing theory to model SDN. He presented several challenges in modeling SDN systems, for instance, the latencies imposed by moving data between different caches, resource contention among threads, interruptions triggered by hardware or software processes, and the implementation design of queues. Another challenge involved is the modeling of the finite capacity of a system and limited buffer size. However, most of the related works, as the author mentioned, have omitted these two challenges and have considered them infinite.

The work in [17] by Jarschel et al. is one of the initial attempts to model the OpenFlow-based interaction between a switch and a controller. The authors provided a performance model based on the queueing theory of an OpenFlow system. They derived some performance parameters in order to introduce an analytical model. To do so, they set up an OpenFlow testbed. They considered a feedback-oriented queueing system model, which was divided into a forwarding queue system, i.e., a switch, and a feedback queue system, i.e., a controller. Their model for the switch was based on a M/M queue and a $M/M-S$ for the feedback queue, meaning the queue size was infinite at the switch as opposed to finite queue size at the controller. Their model has some limitations; for instance, it captured only the interaction between a single node and the controller.

As Duan et al. showed in [9], the average delay relies on the flow table state within switches. They defined the average packet delay as the result of two kinds of delay. One is a delay within the switch if it has to forward to the controller. The other is a delay after the handover decision has been made, as the two switches that are involved in the process have to contact the controller to accomplish the handover. In their work, the authors provided an analytical framework to quantify the total packet delay in an SDN-based network in a centralized controller setting. They studied the single-node model in which they did not capture the handover control messaging exchanged between a switch and the controller.

In [18], the single-node model, where they modeled the feedback interaction between a switch and the controller, is also considered. They used the Jackson network to model the data plane with modifications to fit the nature of the traffic flow in an OpenFlow-based network. The controller is modeled as an external entity based on a M/M queue. They assumed the Poisson distribution for arrival rates at both the switch and the controller and had exponential service rate in both. Their performance measures were the average packets sojourn time and the distribution of time spent by the packet.

In addition to using the queueing theory for evaluating SDN deployment, the authors of [19] provided an analytical modeling based on network calculus theory. They evaluated the SDN controller and switch in terms of delay and queue length. They provided the upper bound of packet processing delay of switches and the upper bound of the controller queue length. However, they did not consider the feedback between the switch and controller. Additionally, there were no simulation data recorded to validate their mathematical model.

Most of the literature works reviewed have modeled the single-node case, meaning they have modeled the interaction between one switch in the data plane and one controller. A downside to the single-node modeling is that it is an unrealistic modeling for such architectures. They omit the possibility of incoming flows from multiple switches, meaning that their input to the switch is inaccurate. To the best of our knowledge, the work in [20] by Mahmood et al. is the first that considered the multiple-node case, where more than one switch exists in the data plane. They extended their work in [18] to model OpenFlow *Packet-in* messages in the multiple node case. The authors presented an analytical model to estimate the packets' sojourn time and the density of the packets that can be pumped into the network. They studied both infinite and finite buffer size scenarios. However, their modeling did not consider the propagation delay. Even though this delay might be small, it is essential to be considered while modeling multiple nodes, where the propagation times may differ as the multiple nodes get placed in different locations. Moreover, another study that took into account modeling several switches interacting with one controller is shown in [14]. Shang et al. studied the impact of *Packet-in* messages on the performance of OpenFlow switches and controller. Additionally, they modeled switches as M/H_2, where the service rate follows a two-phase hyperexponential distribution, whereas the controller is modeled as M/M. They also showed that the performance of the network drops rapidly as the probability of *Packet-in* messages increases. They provided an analytical model; however, no simulation data were recorded.

In all the mentioned studies, the proposed models consider only the case of exchanging one kind of message, *Packet-in*s. However, based on our research direction, we need to incorporate another type of message that is tightly coupled with handover procedures, i.e., *Port-status* messages. Additionally, we need to consider the multiple node case.

2.2. Mobility Management Solutions for LTE. Through analyzing existing LTE architecture, one can find several

proposals focusing on SDN for changing the current cellular network. Nguyen et al. in [21] gave a general view of challenges faced by LTE architecture: it is expensive to modify and upgrade the current system, it lacks the support of new network services and applications, and it generates ineffective management of existing network resources. They also provided a new way to fix the current LTE system by using SDN. The authors provided a potential possibility for changing the current cellular network to SDN and virtualization-based architecture.

Chourasia and Sivalingam in [22] revealed a possible solution for using SDN to replace existing LTE architecture. In this architecture, they proposed the use of a centralized controller that has a global view of the whole network, which makes the resources management easier. Based on their simulation results, this solution has outperformed existing LTE architecture in terms of signaling cost and handover latency. On the contrary, the work in [23] showed that physically centralized SDN has disadvantages; it limits the distribution of handover decisions and has scalability issues.

In [24], the authors discussed the mobility management of future cellular networks. They argued that the mobility management function should be distributed in order to handle an increasing load of data and support scalability of the system. They showcased three approaches for distributing mobility management: a PMIPv6-based approach, a routing-based approach, and an SDN-based approach. The first approach is based on Proxy Mobile IPv6 protocol. The idea here is to distribute mobility access gateways (MAGs), which work close with the local mobility anchor (LMA). The second approach works in a distributed way by removing existing anchors in the system. When users move to new locations, nodes in the system need to calculate new routing tables information. The last approach is based on the SDN concept; the controller in this approach configures rules for switches, which are working as distributed mobility management gateways.

Valtulina et al. in [25] elaborated on an SDN architecture that provides distributed core functions as entities in the network and uses cloud-computing technology to process network data. This architecture provides traffic redirection during the handover and can keep user's data transfer seamless using OpenFlow protocol. Moreover, Braun and Menth in [26] described the operation of OpenFlow and different aspects of SDN-based architecture design. For the control plane, they discussed the controller distribution as well as methods of signaling management.

3. Network Model

The communication between a switch and the controller takes place through the OpenFlow channel where only certain types of messages are standardized [27]. It is a bidirectional channel so that both parties can initiate messages. In OpenFlow v.1.3.1, from a switch to the controller, a switch may initiate either asynchronous messages or symmetric messages. The asynchronous messages include *Packet-in*, *Port-status*, *Flow-Removed*, and *Error* messages. The symmetric messages include *Hello*, *Echo*, and *Experimenter* messages. These messages provide means to the controller to implement all kinds of functions, such as forwarding, filtering, blocking, etc., by configuring forwarding devices' flow tables. Note that we use the words switch, node, and forwarding device, interchangeably referring to the data plane devices, throughout this paper.

Regarding a *Packet-in* message, when a packet arrives at a switch, we have two cases. First, the packet has a flow entry that matches the packet's header, so it is forwarded based on matching fields. Second, the packet has no entry; then a *Packet-in* message is sent to the controller to obtain knowledge. The controller decides on the best forwarding path and updates relevant switches with new flow entries to be installed in their flow tables. Regarding a *Port-status* message, when a mobile node (MN) is disconnected from a switch, a *Port-status* message is initiated by that switch and sent to the controller. Similarly, when an MN is connected to a new switch, another *Port-status* message is initiated by the new switch and sent to the controller. Upon receiving a *Port-status* message, the controller updates some or all of the nodes with up-to-date entries about that MN.

To serve the purpose of our network, i.e., a mobile network, the two main messages that would frequently be sent by a switch are *Packet-in* and *Port-status*. However, the exchange of the rest of the messages is relatively small as they are either initiated upon the start up of the network or for testing purposes. Therefore, we considered analyzing the behavior of *Packet-in* and *Port-status* messages combined in our system.

We consider a network where hosts are connected to n OpenFlow switches. All of the OpenFlow switches are controlled by a centralized controller, C, which resides in a relatively remote location and has a global view. Our network is a typical OpenFlow-based network, where *Packet-in* messages are considered a cornerstone of its implementation.

In our network, any host can belong to one of two groups at time t. A group of hosts is connected to a switch, s_i, without attempting to handover while the other group is switching from one switch, s_i, to another, s_j. We refer to the hosts of the second group as mobile nodes (MNs). An MN handover requires two connection requests to the controller, one from its old switch and the other from its new switch, *off-port* and *on-port*, respectively. The movement of the MN from one location to another requires a clear configuration of all or multiple switches (i.e., sending out *Flow-mod* messages by C). Figure 1 gives a visual illustration of the exchange of messages between the switches and controller triggered by the mobility of a node.

In this section, we aim at quantifying the handover delay incurred due to the propagation of control messages between switches and controller as well as for processing and queueing those messages at the switches and controller.

3.1. Assumptions. We assume that every *off-port* message corresponds to an *on-port* message. Intuitively, the number of *off-port* messages equals the number of *on-port* messages; therefore, we can say that both kinds of control messages can

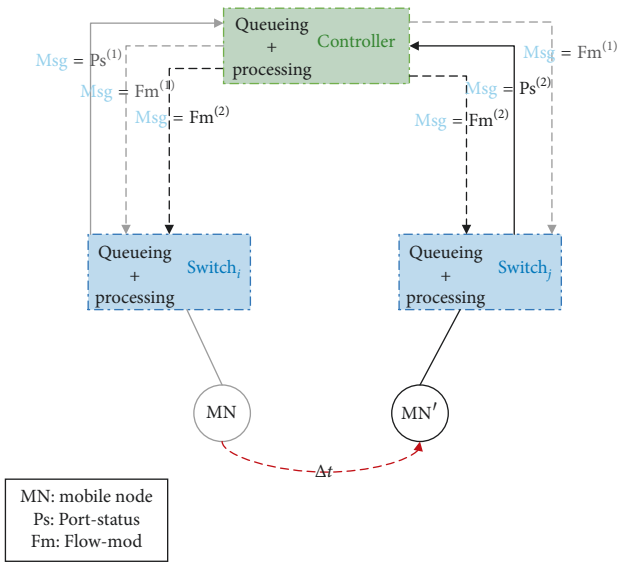


Figure 1: The exchange of messages between the switches and controller upon mobility.

be represented as *Port-status* message. We also assume that the controller has infinite buffer size, which means no packets are dropped. It is debatable whether or not this assumption can be adopted because, usually, controllers are equipped with huge resources that are virtually infinite but actually limited. However, for simplicity, we assume infinite queue size at the controller. We additionally assume infinite switches' queues in line with previous works [9, 13, 17, 28], as studying the impact of limited buffers and dropping packets rates are out of the scope of our analysis.

Regarding the handover calls distribution, the authors of [29, 30] have shown that they can be modeled as Poisson. Convinced by their validated results, we assumed that the occurrences of *Port-status* events follow the Poisson distribution. To put it another way, we argue that since the arrival rate of *Port-status* events are human-initiated, the interarrival and service times are independent and exponentially distributed and features the memoryless property. Additionally, in our modeling, we assumed that the service rates of all nodes were load-independent.

3.2. Model Description. In this section, we start with modeling the *Port-status* messages; then we expand our model to include other types of messages that are essential in SDN-based networks, *Packet-in* message.

3.2.1. Modeling Port-Status Message. Regarding handover, we consider analyzing the hard handover, i.e., break-before-make approach. Our analysis focuses on the process that follows a handover decision. When an MN is disconnected from one switch, say s_i, it gets attached to another switch, say s_j, where $i, j \in \{1, 2, \ldots, n\}$. This process triggers two asynchronous *Port-status* messages to update the controller of MN's movement. On the controller side, its network map has to be in an up-to-date state. Accordingly, the controller initiates *Flow-mod* messages to all or some of the switches changing the flow entries of that MN.

A *Port-status* message generated by a switch can be either *off-port* or *on-port* message. Let us intuitively assume that the number of *off-port* messages in our network equals the number of *on-port* messages. Both types of *Port-status* messages are treated the same, but, of course, the controller can distinguish between them; therefore, we refer to both kinds of messages as one, *Port-status*.

In part, every *Port-status* message triggers three other messages: *Flow-mod* messages by C, another *Port-status* message by the new switch, and a second *Flow-mod* message by C upon receiving the new switch's message.

Based on the principles of modeling using the queueing theory, there are a set of parameters that need to be identified, including the arrival rate of requests, service time, the capacity of the system, the size of the source, and the service disciplines [31]. Taking into consideration our assumptions and using Kendall's notations [32] for queueing systems, the switches and the controller are modeled as M/M systems, where we assume that the queues are infinite and the service discipline is FIFO. We considered having an open queueing network where events can enter the system from outside and can also depart the network from any node. We additionally assumed a Markovian system, where a system's state is defined by the number of events (i.e., jobs) at nodes. If k_i is considered as the number of jobs at node i, then the state of the network is defined as $(k_1, k_2, \ldots, k_n)$. Note that throughout this paper, we use the words jobs and events interchangeably to refer to the buffered elements that need to be processed in a queue.

Now, let us assume that the arrival rate of *Port-status* messages initiated by a node i is λ_{ih}. Then, the flow of *Port-status* events captured by any node i is modeled as

$$\gamma_{ih} = \lambda_{ih}. \tag{1}$$

Note that a port can change from *on* to *off*, or a new port gets attached. Either way, the two events are treated alike.

Accordingly, Γ_{ih} is the total net arrival rate of port changes from the whole network to node i (Figure 2) and is given as follows:

$$\Gamma_{ih} = \gamma_{ih} + \sum_{j=1, j\neq i}^{n} \left(\lambda_{jh} \times \nu_j^i\right), \tag{2}$$

where $\nu_j^i \in \{0, 1\}$ is an indicator applied to model the effect of the *Port-status* message after being handled by the controller. If i's flow table has to be configured based on j's *Port-status* message, then $\nu_j^i = 1$; otherwise, $\nu_j^i = 0$. Note that some or all nodes may be configured by the controller upon receiving a *Port-status* message. Regarding the controller, Γ_{ch} is the total net arrival rate of *Port-status* messages at C and is given as follows:

$$\Gamma_{ch} = \sum_{i=1}^{n} \lambda_{ih}. \tag{3}$$

So far, our modeling includes only *Port-status* messages, meaning that we consider a single-class queueing network.

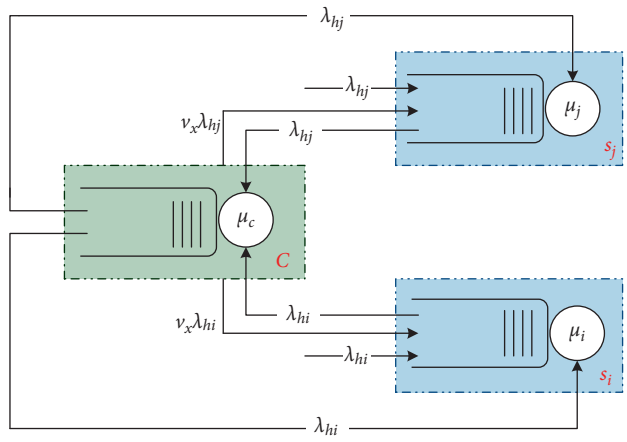


Figure 2: A queueing model of *Port-status* messages.

However, our model is extended in the following section and includes another type of message, *Packet-in*.

3.2.2. Multiclass Open Network Modeling. When we model a network of queues where all the jobs are of the same type with regard to their service times and routing probabilities, then this is called a *single-class network*. If the network model, however, is extended to include multiple job classes, then it is called a *multiclass network*. In this extended model, different types of traffic can be combined in an open or closed setting [33]. In our case, so far, we have considered modeling the *Port-status* messages in a network, which is unrealistic. In any SDN-based network, *Packet-in* messages are essential. As has been shown by [27], other types of messages are exchanged between the data plane and the control plane for different purposes. However, in a mobile network, the most common two messages are *Packet-in* and *Port-status*. These two messages are treated differently, as illustrated in the flow chart in Figure 3. Therefore, our network is modeled as a *multiclass network*, where we restrict our analysis to the two types of messages.

To our queueing network that contains two job classes, we need to define the following symbols:

(i) R is the number of job classes, where in our network $R = 2$.

(ii) k_{ir} is the number of jobs of class r at node i, let $r = h$ if the intended class is *Port-status*, and $r = p$ if the class is *Packet-in*. Similarly, k_{cr} is the number of jobs of class r at the controller.

(iii) K is the number of jobs in different classes.

(iv) S_i is the state of node i, where $S_i = (k_{ih}, k_{ip})$. Similarly, S_c represents the state of C.

(v) S is the overall state of the network with multiple jobs where the state probability is represented by $\pi(S_1, S_2, \ldots, S_n)$. Note that the sum of the probabilities of all states should be one since we assume stability in the system.

(vi) μ_{ir} is the service rate of node i for the job class r. In our network, we have μ_{ip}, which represents the service rate at a switch for *Packet-in* messages, while μ_{ih} is the service rate for *Port-status* messages. In our model, we assume $\mu_{ip} = \mu_{ih}$. Similarly, μ_{cr} is used to represent the service rates for the two messages at C.

(vii) λ_{ir} is the arrival rate of job class r at node i, so λ_{ir} is either λ_{ip} or λ_{ih}.

(viii) Γ_{ir} and Γ_{cr} represent the traffic of class r at node i and C, respectively, and they are both functions of λ_{ir} and λ_{ip}.

So far, we have defined the parameters of modeling the *Port-status* message in the previous section. Now, we define *Packet-in* message parameters in addition to the aforementioned parameters. Let the arrival rate of packets, in general, to a node i be λ_{ip}. Then, the probability of a *Packet-in* message to be sent to the controller from node i is q_i^{nf}, which means there is no flow entry regarding that packet. On the contrary, in case a switch matches an entry for incoming packets, our modeling should include the traffic between nodes. Therefore, the probability between any two nodes i and j is p_{ij}. At any node i, the flow of input packets can be modeled as

$$\gamma_{ip} = \lambda_{ip} + \sum_{j=1, j\neq i}^{n} p_{ij}\gamma_{jp}. \tag{4}$$

Then, the total net arrival rate of packets, excluding *Port-status* messages, is given as

$$\Gamma_{ip} = \gamma_{ip} + q_i^{nf}\lambda_{ip} + \sum_{j=1, j\neq i}^{n} \left(q_j^{nf} \times u_j^i\right)\lambda_{jp}, \tag{5}$$

where $u_j^i \in \{0, 1\}$ is an indicator of whether a flow has to be routed between i and j or not, as used in [20]. Using this indicator, the updates the controller sends to all or a subset of the nodes upon receiving *Packet-in* are included. Accordingly, the arrival rate of *Packet-in* messages to the controller is given as

$$\Gamma_{cp} = \sum_{i=1}^{n} q_i^{nf}\lambda_{ip}. \tag{6}$$

3.3. Limitations. Here, we list the limitations of our model. Firstly, we modeled the controller and switches as a single queue and not per interface. Also, we did not consider the case of dropped packets at either the controller or switches due to overload, which goes in line with our assumptions that we have infinite queue sizes.

OpenFlow protocol has different message types to be exchanged between a switch and the controller, and in our model, we only focused on two types of messages that are the most frequent messages in the networks we studied, i.e., *Port-status* and *Packet-in* messages. Moreover, we assumed TCP traffic only, meaning only the header of the first packet of each new flow is sent to the controller, in contrast to UDP traffic, where the incoming packets of a new flow are relayed to the controller until the associated flow entry is installed.

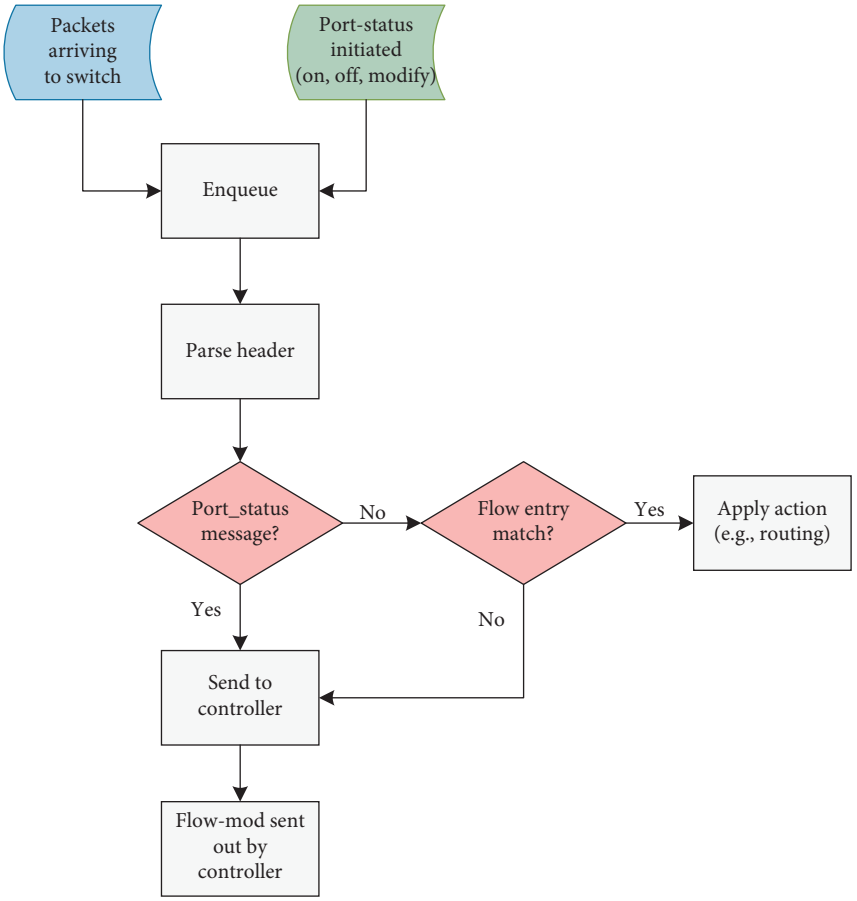


Figure 3: A flow chart of an OpenFlow switch handling two messages, *Packet-in* and *Port-status*.

4. Performance Measures

In this section, based on our model, the goal is to quantify the handover delay that occurs as a result of exchanging OpenFlow-related messages. It is imperative to distinguish between three main delays: waiting time, sojourn time, and total delay. The waiting time is the time an event spends in a queue at node i with respect to class r. However, the sojourn time, T_{ir}, is the time an event spends at node i, including the time it is being serviced. Regarding our main metric, the total time, D_{tot}, is the time an MN experiences before completing the handover procedure as it will be discussed in further detail in this section.

Let W_i^c be the time of interaction between node i and C upon a *Port-status* event, triggered by s_i. Then, W_i^c can be defined as

$$W_i^c = T_{ih} + T_{i(\text{prop})}^c + \max\left\{T_{x(\text{prop})}^c\right\} + T_{ch}. \tag{7}$$

Note that $T_{i(\text{prop})}^c$ is the time for propagating a *Port-status* message from s_i to C, given that the T_{prop} between the controller and all switches are assumed to have the same link parameters. In OpenFlow design, the controller reacts to a *Port-status* message by sending out a *Flow-mod* message to all or a subset of switches in parallel. In other words, C sends reactive *Flow-mod* message(s) to switches in parallel, which may take roughly the propagation time required to reach the furthest switch; hence, we define the maximum propagation time as

$$\max\left\{T_{x(\text{prop})}^c\right\} \text{ where } x \in \{1, 2, \ldots, n\} \text{ and } x \neq i. \tag{8}$$

Then, the expected value of W_i^c can be further written as

$$E\left[W_i^c\right] = E\left[T_{ih}\right] + T_{i(\text{prop})}^c + \max\left\{T_{x(\text{prop})}^c\right\} + E\left[T_{\text{ch}}\right]. \tag{9}$$

Based on the above explanation of how the controller handles *Port-status* messages, we infer that to complete the handover procedure of an MN, the total delay experienced by that MN is

$$D_{\text{tot}} = W_i^c + W_j^c, \tag{10}$$

where W_i^c represents the time of an *off-port* message handled by an old switch s_i, and W_j^c represents the time of an *on-port* message handled by a new switch s_j. This yields to

$$D_{\text{tot}} = E\left[T_{ih}\right] + T_{\text{prop}} + E\left[T_{jh}\right] + E\left[T_{ch}\right] + E\left[T_{ch}^{(2)}\right], \tag{11}$$

where

$$T_{\text{prop}} = T_{i(\text{prop})}^c + \max\left\{T_{x(\text{prop})}^c\right\} + T_{j(\text{prop})}^c + \max\left\{T_{y(\text{prop})}^c\right\} \quad \text{and } x \neq i, y \neq j. \tag{12}$$

Note that $T_{ch}^{(2)}$ represents the sojourn time of an *on-port* message at the controller. We distinguish the two times because they are independent.

To find the mean response time of a *Port-status* event, we use the Mean Value Analysis (MVA) solution. MVA has been introduced as an iterative technique to obtain an exact solution for some performance measures including sojourn times in separable queueing networks. However, the MVA's original version was proposed for closed networks only. In 1981, Zahorjan and Wong in [15] presented the MVA solutions for open and mixed networks. Note that we are in line with their assumptions, which are the existence of a FIFO queueing discipline and a single server. In this section, we use their findings to compute our performance measures of interest. The MVA for open queueing networks depends on two theorems, Little's theorem [34] and the arrival instant theorem [35]. Therefore, we consider the Poisson Arrivals See Time Averages (PASTA) property of Poisson arrivals, which indicates that in a network in statistical equilibrium, the time averages equal the arrival averages [36]; hence, the sojourn time can be formulated as

$$E[T_{ih}] = \frac{1}{\mu_i}\left(1 + \left(E\left[K_{ip}\right] + E\left[K_{ih}\right]\right)\right). \tag{13}$$

As proven by [15], for any two classes, in our case h and p, the following relation is satisfied:

$$E[K_{ih}] = \frac{\rho_{ih}}{\rho_{ip}} E\left[K_{ip}\right], \tag{14}$$

where ρ_{ih} and ρ_{ip} are the utilization of node i with respect to *Port-status* and *Packet-in* messages, respectively. As we mentioned in our model description, the utilization is load-independent.

With the help of Little's theory, $E[K] = \lambda.E[T]$, and by substituting it in the previous equations, we get

$$E\left[K_{ih}\right] = \frac{\rho_{ih}}{1 - \left(\rho_{ip} + \rho_{ih}\right)}. \tag{15}$$

Similarly, the previous equation is applied to find the controller's response time regarding *Port-status* messages.

So far, based on our assumption, and the findings and theories of previous works, we formulated the expected response time of a handover procedure in an OpenFlow-based network. However, there is a constraint that needs to be satisfied for our model to work. The utilization of queues has to be less than unity. Let ρ_i and ρ_c represent the total utilization of switch i and controller C, respectively; then, $\rho_i < 1$ and $\rho_c < 1$, given the following:

$$\rho_i = \rho_{ih} + \rho_{ip}. \tag{16}$$

Similarly, ρ_c is defined.

5. Proposed Approach

As authors in [9] showed, an SDN-based solution is more suitable for "less-sensitive" to latency applications. However, mobility management applications are latency-sensitive. In fact, there are two salient factors that contribute to increased latency, transmission delay, and processing delay [7]. Therefore, we need to configure an SDN-based solution that minimizes handover latency in terms of transmission and processing delays.

5.1. Main Idea. In our approach, we aim at targeting the two aforementioned contributors. Firstly, we need to minimize processing latency, and to do so, we propose offloading handover handling to dedicated entities that are separate from our controller and switches. We call these entities Mobility Handling Entity (MHE). Secondly, we aim at minimizing transmission latency by placing the MHE physically closer to switches. Essentially, by offloading the burden of handling *Port-status* messages to entities other than the controller, we efficiently minimize latency. In short, the role of each element in our solution is as follows:

(i) Controller: it maintains the information of all devices in the network, MHEs, OpenFlow switches, etc. Mainly, it has the global view.

(ii) Mobility Handling Entity (MHE): MHEs are placed physically close to switches and handle handover procedures asynchronously with the controller.

(iii) OpenFlow switch: switches focus on keeping data flowing from the source to the target. They are managed by the controller through the OpenFlow protocol. They exchange commands with the controller to perform several tasks, such as updating the flow table, triggering *Port-status* messages, and other stats-related messages.

In our solution, the functionality of handling handover is installed on MHEs to allow them to work asynchronously with the controller. In this case, the controller has to maintain specific information about MHEs in the network. Each MHE is associated with a table of four columns, MHE_ID, MHE_IP, Status, and Update_Time. MHE_ID and MHE_IP are used to locate a specific MHE. The Status is used to indicate whether that MHE is "active" or "inactive." Lastly, the Update_Time marks the time an MHE gets updated by the controller. As for the switches, we assume that they exchange handover-related messages with MHEs through OpenFlow channels. Therefore, in our approach, MHEs are considered SDN-controllers with limited functionality and a specific task.

5.2. Proposed EPC Controller Architecture. In this section, we apply the proposed solution to a cellular network where mobile nodes are connected to eNodeBs, assuming that every eNodeB is connected to an OpenFlow switch. All OpenFlow switches are controlled by a centralized controller that resides in a relatively remote location and, so, has a global view; thus, it governs the handover procedure. Additionally, we consider a hard handover, which is in practice in LTE systems. Our preliminary work on this setting has been published in [37].

As previously stated, the handover signaling latency can be reduced by minimizing the processing and transmission

delays of managing entities [7]. Therefore, we presented our proposed solution in the LTE setting, where we assigned entities that have to be responsible for handover messages. Thus, we distributed the functionality of the EPC controller. Our proposed EPC controller architecture enables multiple functional entities to work in conjunction with the controller, which operate as MME and other handover-related functions (Figure 4). With proper routing information from the controller, these entities can work asynchronously to handle handover in the network. Using these functional entities, we can potentially reduce the workload of the controller and make the handover process more efficient and the system more scalable.

Different from the centralized EPC controller architecture proposed by [22], our proposed EPC controller architecture divides the traditional controller into a basic controller and multiple MHEs. Hence, the control plane functionality is not exclusive to the controller. MHEs serve functions such as MME, S-GW, and P-GW combined. They are placed physically close to eNodeBs and perform handover procedures asynchronously with the controller. The protocol of our proposed solution is illustrated in Figure 5(b) as opposed to the protocol proposed by [22] in Figure 5(a).

6. Numerical Results and Analysis

In this section, we divide our analysis into four directions. Firstly, we verify and validate our analytical model by comparing its results to a conducted simulation experiment output. Secondly, we evaluate and analyze our proposed solution in more detail. Thirdly, we summarize our results published in [37]. Lastly, we discuss some implementation issues in SDN-based LTE systems that need to be addressed.

6.1. Verifying Analytical Model. An experiment was conducted to compare the simulation results to our analytical model results. We have developed a discrete event simulator using Matlab, and our simulation parameters are listed in Table 1. For the value of q^{nf}, Jarschel et al. in [17] showed that in a production network, the probability of *Packet-in* messages is 4%. We also borrowed some of the measurements in [17], such as the service rates of controller and switches. The size of OpenFlow messages are mentioned in [27]. We needed the sizes to determine the transmission and total delays based on our assumptions of the link type and speed, as listed below.

For the arrival rates of *Port-status* and *Packet-in*, we used the aforementioned values and substituted them in the following equations.

$$\rho_c = \frac{\Gamma_{cp} + \Gamma_{ch}}{\mu_c}, \tag{17}$$

$$\rho_i = \frac{\Gamma_{ip} + \Gamma_{ih}}{\mu_i}. \tag{18}$$

By equating Equations (17) and (18) at load $\simeq 1$, we got values of λ_{ip} and λ_{ih}. Note that λ_{ih} has to be much smaller than λ_{ip} in practice.

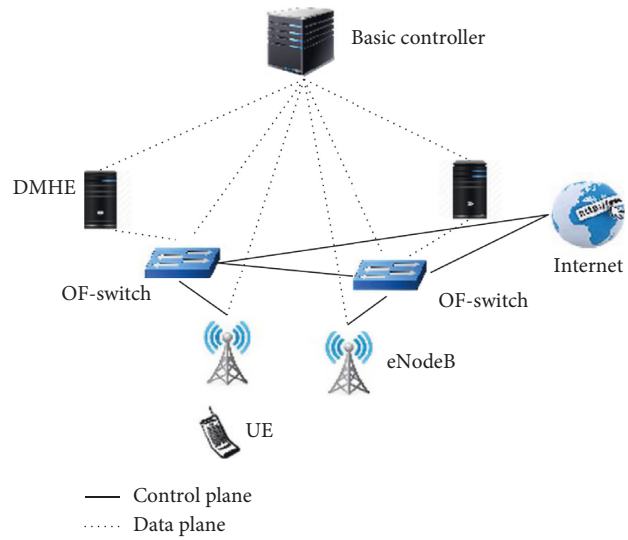


Figure 4: Proposed EPC controller architecture.

Our comparison of the simulation results and the analytical computations shows similar trends that start and end almost the same, as depicted in Figure 6. Additionally, they both show a rapid increase in delay around load = 0.7, which is expected. However, the divergence that occurs around load = 0.3 can be contributed to the simulation running time and/or hardware specifications.

After gaining confidence in our simulation setup, we carry out our analysis in the following section.

6.2. Evaluating Proposed Approach. To show the difference that an MHE causes, we simulated the response time *Port-status* messages experience in two approaches: a network of queues excluding MHE and a network of queues including MHE. Note that our network of queues consists of one switch, one controller, and an offloading entity in the second topology. Initially, we compared the two topologies when both MHE and *C* have the same processing time, which is considered the worst-case scenario regarding MHE's service rate. As depicted in Figure 7, the response time drops tremendously in the case where MHE handles the *Port-status* messages.

MHEs are entities assigned by the network designer to handle *Port-status* messages separately from the controller for the sole purpose of decreasing the time that a handover procedure may take. Those entities can be designed to be separate queues in the controller or can be separate physical devices that can be placed anywhere in between the controller and the switches. Additionally, they can be given different service rates to indicate different capabilities. Therefore, we break down our analysis in the following part into the impact of different service rates and the impact of physical placement.

6.2.1. Impact of Different Service Rates. MHEs' design is determined by the network operator. In this part, we try to give different options and analyze their impacts.

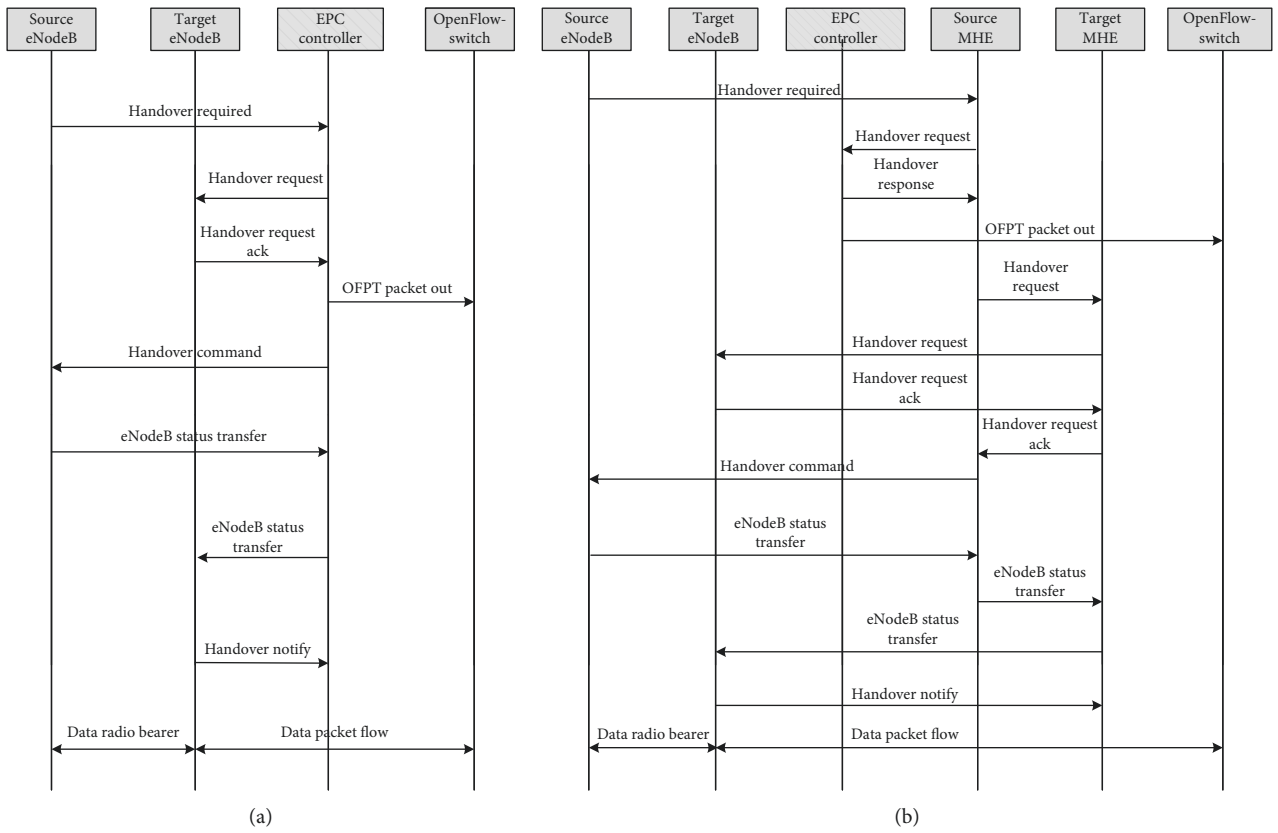


Figure 5: Handover procedure flow in two different architectures. (a) Centralized EPC controller architecture [22]. (b) Proposed EPC controller architecture.

Table 1: Simulation parameters.

Parameter	Value
Probability to send *Packet-in* to controller q^{nf}	0.04 (4%)
Average service time of the controller	= 240 μs
Average service time of a switch	= L9.8 μs
Size of *Packet-in* message	128 B
Size of *Port-status* message	128 B
Size of *Flow-mod* message	128 B
Distance between switches and controller	1–5 kms
Links between switches and controller	Optical
Link speed	1 Gbps

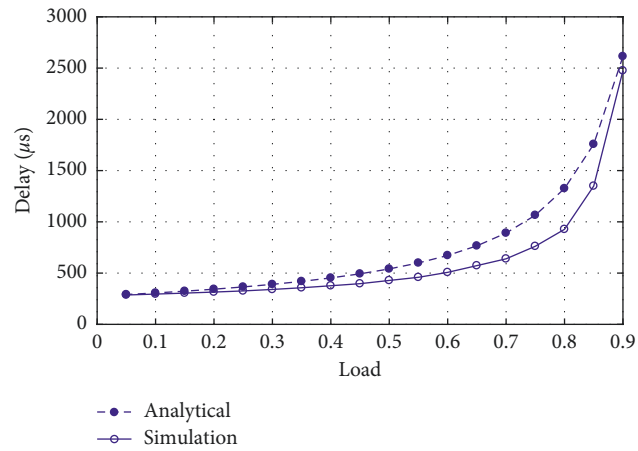


Figure 6: The simulation and analytical model results.

As a matter of fact, the service rate plays an important role in queues' processing times. As we are adding MHEs, we have the freedom to choose the service rate. However, we will compare three service time values, 9.8 μs, 240 μs, and 125 μs. We deliberately chose those three values. We wanted to see the impact of having an MHE with a service time as fast as that of the switch (i.e., 9.8 μs). Then, we studied the impact of having a service time as slow as that of the controller (i.e., 240 μs). Lastly, we suggested a service time value that is in between the two aforementioned values (i.e., 125 μs). Intuitively, the impact is quite obvious, as depicted in Figure 8. Based on the purpose and resources that are available, the network designer has to determine a suitable service rate.

6.2.2. Impact of Physical Placement. In networking systems, delivery time can be broken down into transmission time and propagation delay. The transmission delay represents the time from the beginning until the end of a message transmission, so it is correlated with the packet size and the bit rate of the medium. The propagation delay is the time the first bit takes to travel from a source to a destination, and therefore, it depends on the physical medium as well as the distance separating the correspondents. In our analysis, we

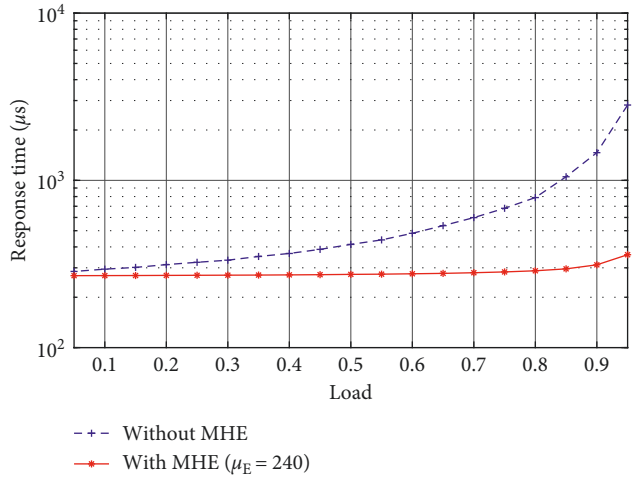


Figure 7: Comparing *Port-status* response time of a model with MHE to a model without MHE.

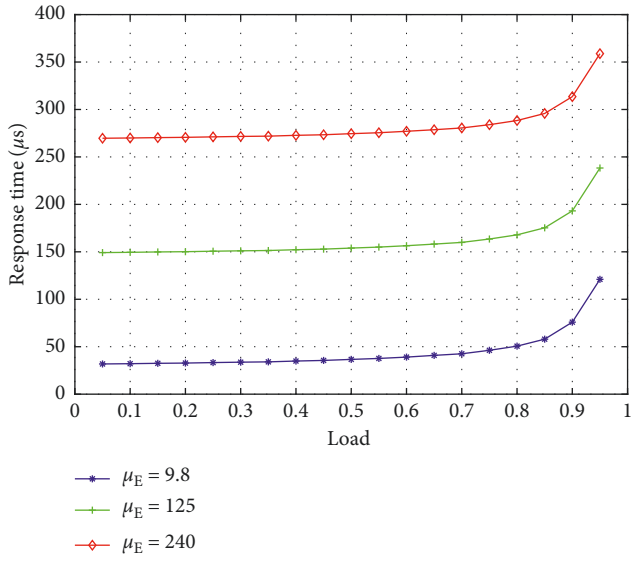


Figure 8: Response time of MHE with different service rates.

assumed 1 Gbps optical links over 1 to 5 km distance. We found that in this setting, the difference between 1 km and 5 km is a matter of microseconds. Therefore, we argue that the location of MHEs in our setup does not have much impact on the total delay, so the entities can be placed in convenient locations whether within the controller unit or physically separate in a particular place. However, using different link parameters may have an apparent impact on transmission latency.

6.3. Preliminary Results. We conducted a comparative analysis between the centralized EPC controller [22] and our proposed EPC controller architectures [37]. This was undertaken with the widely used discrete event simulator NS3 [38]. In our experiment, we utilized the available cellular network modules and OpenFlow to simulate LTE functions in SDN architecture. By analyzing the handover latency and average throughput per user, we aimed at showing the performance of these two different architectures under certain circumstances.

6.3.1. Handover Latency. The handover latency measures the time that elapses from the time the source eNodeB sends the handover request to the controller or to another handover-handling entity until it receives the handover notification from target eNodeB. In order to achieve an overall comparison of the two architectures in terms of the handover latency, we generated UDP packets for each link from one end to the other with different occupancy percentages of background traffic to simulate a real-life environment. We took three scenarios into consideration: the idle network as 30% of background traffic, the standard network as 50%, and the busy network as 70%. In each simulation, we made mobile nodes handover from source eNodeB to target eNodeB and generated the sum of the latency of all MNs. As shown in Figure 9, for both architectures, the handover latency increased along with the growing background traffic from 30% to 70%, whereas the proposed EPC controller architecture continuously outperformed the centralized EPC controller architecture with smaller handover latency under the same background traffic. In standard traffic conditions, our architecture experienced 20% less handover latency than the centralized architecture. However, in busy networks (i.e., 70%), the performance of both architectures degraded in terms of handover latency. Regardless, these results indicate that our proposed EPC controller architecture has better performance when dealing with handover under the same circumstances.

6.3.2. Average Throughput per User. The average throughput is a critically important metric that measures the system usability and scalability. It can be calculated using the following formula:

$$V_T = \frac{B}{(T \cdot n)}, \tag{19}$$

where V_T is the average throughput per user, B represents the total bytes successfully delivered, while T represents the time that elapses during the reception of the data, and n represents the number of users in the system. We set 20% of the users to perform handover while simulating a real-life environment. Each MN sent 5 packets per second to a remote host, and the size of each packet was 38 bytes. After recording the data, Equation (19) is applied to calculate the average throughput per user. As shown in Figure 10, for the same number of users in each system, we can see that our proposed EPC controller architecture had greater value on throughput over centralized EPC controller architecture. With a growing number of users, the average throughput for proposed EPC controller architecture decreased as it did for the centralized EPC controller architecture. Nonetheless, our proposed EPC still outperformed the centralized EPC under each user category. These results indicate that the proposed EPC controller architecture can provide better data service for each user under the same conditions.

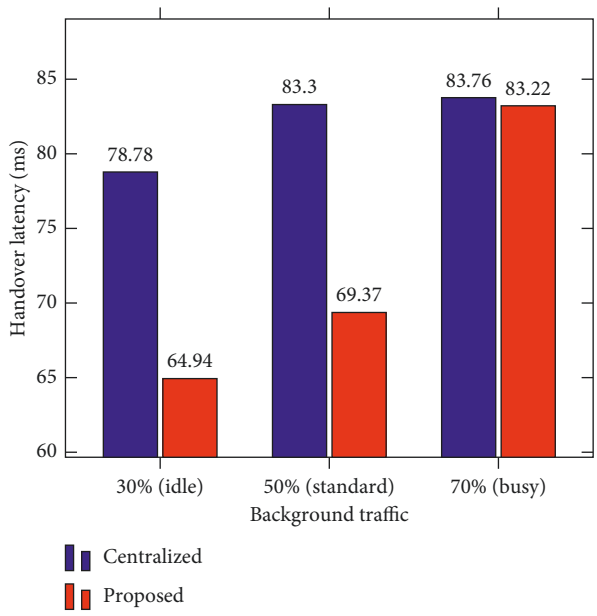


Figure 9: Handover latency under different traffic conditions.

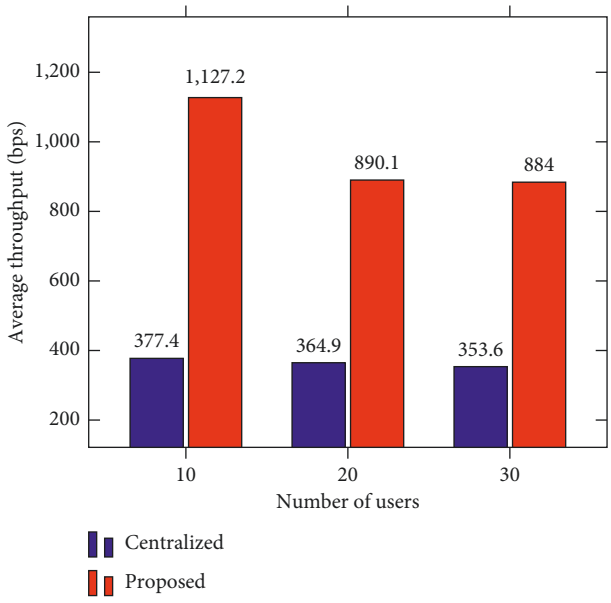


Figure 10: Average throughput per user.

In part, the proposed functionality-distributed EPC controller architecture sacrifices the general view held by the MHEs and may potentially increase the operational cost. However, it has distinct advantages in the areas of handover latency and carries a higher average throughput per user when compared to the centralized EPC controller architecture. As a result, this is a better option for service providers who prefer low handover latency and a high average throughput per user of the system.

6.4. Implementation Issues. According to the specifications of OpenFlow v.1.3.1, a switch initiates a *Port-status* message if a port was added, a port was removed, or an attribute of a port has changed. In our case, the switches have to send *Port-status* messages upon the connection/disconnection of mobile nodes. Note that, in our system, MNs are connected to eNodeBs and not directly connected to the OpenFlow switches. Therefore, the current deployment of SDN-based LTE systems should be modified to enable direct linking between MNs' mobility and the switches. To overcome this challenge, one option, which we assumed in our simulation, is to embed the OpenFlow switches in the eNodeBs to make the OpenFlow switch's interfaces connected directly to MNs. Another option is to make adaptations and modifications to the OpenFlow protocol. For this option, the modification should allow interpreting connection/disconnection alerts from eNodeBs into *Port-status* messages in the switches.

7. Conclusion

It is important to model the OpenFlow controller to switch interaction in terms of the two kinds of messages that are commonly used in mobile networks. Indeed, this modeling helps us better understand the underlying causes of handover delay and then helps us propose effective methods to minimize it. In this model, we have modeled two OpenFlow messages: *Packet-in* and *Port-status* in a multiclass open network. We have modeled each message independently since they are two different traffic and need to be treated differently by the controller. Our aim has been to quantify the handover delay incurred due to queueing, processing, and propagating handover control messaging between switches and the controller. Then, we have proposed offloading the mission of handling *Port-status* messages to separate entities in order to overcome some of the shortcomings of the SDN paradigm. Our solution has been validated and evaluated in simulation and applied to an LTE system. We have performed a comparative analysis of a centralized EPC controller architecture and our proposed functionality-distributed EPC controller architecture in the LTE setting. We then have studied two main metrics, handover latency and average throughput per user. Simulations have shown that the proposed EPC controller architecture has better performance in both metrics as compared to the centralized approach, under the same network conditions. Therefore, we argue that the distributed mobility management approach can benefit the handover handling.

In effect, OpenFlow-based SDN and mobile networks as well as distributed mobility management approaches have been extensively discussed in the past but so far not materialized into actual 3GPP networks. SDN is assumed to be applied for the backhaul services rather than directly involved in the operation of a mobile network. In this paper, we have tried to fill in that gap and propose a solution that increases handover efficiency, especially in highly dynamic networks.

For future work, we will tackle some of the limitations mentioned in our queueing model and attempt to improve the proposed solution. Moreover, we plan to extend our

work to study other forms of distributing mobility management such as the multicontroller scenario.

Data Availability

The simulation data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

References

[1] ONF Solution, *Openflow-Enabled Mobile and Wireless Networks*, white paper, 2013.

[2] N. McKeown, T. Anderson, H. Balakrishnan et al., "OpenFlow," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[3] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: past, present, and future of programmable networks," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.

[4] Open Networking Foundation, https://www.opennetworking.org/sdn-resources/openflow.

[5] Cisco Blogs, https://blogs.cisco.com/news/aci-anywhere.

[6] Cisco visual networking index: Global mobile data traffic forecast update, 2016–2021, White Paper, 2017.

[7] C. Marquezan, X. An, Z. Despotovic, R. Khalili, and A. Hecker, "Identifying latency factors in sdn-based mobile core networks," in *Proceedings of Symposium on Computers and Communication*, pp. 484–491, Messina, Italy, June 2016.

[8] K. Tantayakul, R. Dhaou, and B. Paillassa, "Impact of sdn on mobility management," in *Proceedings of 30th International Advanced Information Networking and Applications Conference*, pp. 260–265, Crans-Montana, Switzerland, March 2016.

[9] X. Duan, A. Akhtar, and X. Wang, "Software-defined networking-based resource management: data offloading with load balancing in 5g hetnet," *EURASIP Journal on Wireless Communications and Networking*, vol. 2015, no. 1, p. 181, 2015.

[10] H. Farhady, H. Lee, and A. Nakao, "Software-defined networking: a survey," *Computer Networks*, vol. 81, pp. 79–95, 2015.

[11] N. Feamster, J. Rexford, and E. Zegura, "The road to sdn: an intellectual history of programmable networks," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 2, pp. 87–98, 2014.

[12] D. Kreutz, F. Ramos, P. Verissimo, C. Rothenberg et al., "Software-defined networking: a comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[13] B. Xiong, K. Yang, J. Zhao, W. Li, and K. Li, "Performance evaluation of openflow-based software-defined networks based on queueing model," *Computer Networks*, vol. 102, pp. 172–185, 2016.

[14] Z. Shang and K. Wolter, "Delay evaluation of openflow network based on queueing model," http://arxiv.org/abs/1608.06491, 2016.

[15] J. Zahorjan and E. Wong, "The solution of separable queueing network models using mean value analysis," in *Proceedings of ACM SIGMETRICS Performance Evaluation Review*, pp. 80–85, Las Vegas, NV, USA, September 1981.

[16] C. Thieme, "Challenges for modelling of software-based packet processing in commodity-hardware using queueing theory," *Network*, vol. 49, 2017.

[17] M. Jarschel, S. Oechsner, D. Schlosser et al., "Modeling and performance evaluation of an openflow architecture," in *Proceedings of The 23rd International Teletraffic Congress*, pp. 1–7, San Francisco, CA, USA, September 2011.

[18] K. Mahmood, A. Chilwan, O. Østerbø, and M. Jarschel, "On the modeling of openflow-based sdns: the single node case," http://arxiv.org/abs/1411.4733, 2014.

[19] S. Azodolmolky, R. Nejabati, M. Pazouki et al., "An analytical model for software defined networking: a network calculus-based approach," in *Proceedings of Global Communications Conference*, pp. 1397–1402, Atlanta, GA, USA, December 2013.

[20] K. Mahmood, A. Chilwan, O. Østerbø, and M. Jarchel, "Modelling of openflow-based software-defined networks: the multiple node case," *IET Networks*, vol. 4, no. 5, pp. 278–284, 2015.

[21] V. Nguyen, T. Do, and Y. Kim, "Sdn and virtualization-based lte mobile network architectures: a comprehensive survey," *Wireless Personal Communications*, vol. 86, no. 3, pp. 1401–1438, 2016.

[22] S. Chourasia and K. Sivalingam, "Sdn-based evolved packet core architecture for efficient user mobility support," in *Proceedings of first IEEE Network Softwarization Conference*, pp. 1–5, London, UK, 2015.

[23] S. Kukliński, Y. Li, and K. Dinh, "Handover management in sdn-based mobile networks," in *Proceedings of Global Communications Conference Workshops*, pp. 194–200, Austin, TX, USA, December 2014.

[24] F. Giust, L. Cominardi, and C. Bernardos, "Distributed mobility management for future 5g networks: overview and analysis of existing approaches," *IEEE Communications Magazine*, vol. 53, no. 1, pp. 142–149, 2015.

[25] L. Valtulina, M. Karimzadeh, G. Karagiannis, G. Heijenk, and A. Pras, "Performance evaluation of a sdn/openflow-based distributed mobility management (dmm) approach in virtualized lte systems," in *Proceedings of Global Communications Conference Workshops*, pp. 18–23, Austin, TX, USA, December 2014.

[26] W. Braun and M. Menth, "Software-defined networking using openflow: protocols, applications and architectural design choices," *Future Internet*, vol. 6, no. 2, pp. 302–336, 2014.

[27] Open Networking Foundation, "Openflow switch specification version 1.3.1," Tech. Rep., 2012.

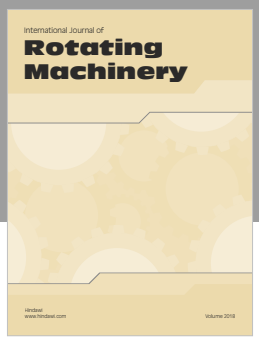
[28] L. Yao, P. Hong, and W. Zhou, "Evaluating the controller capacity in software defined networking," in *Proceedings of 23rd International Computer Communication and Networks Conference*, pp. 1–6, Shanghai, China, August 2014.

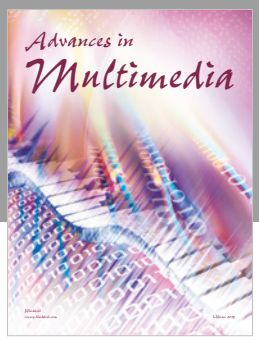
[29] Y. Kirsal and O. Gemikonakli, "Performability modelling of handoff in wireless cellular networks with channel failures and recovery," in *Proceedings of 11th International Computer Modelling and Simulation Conference*, pp. 544–547, Melbourne, Australia, January 2009.

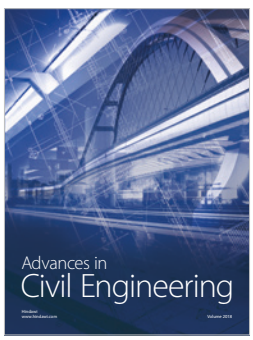
[30] E. Chlebus and W. Ludwin, "Is handoff traffic really poissonian?," in *Proceedings of Fourth International Universal Personal Communications Conference*, pp. 348–353, Tokyo, Japan, November 1995.

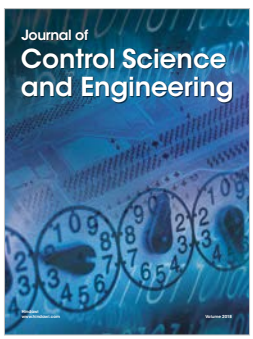
[31] J. Sztrik, *Basic Queueing Theory: Foundations of System Performance Modeling*, GlobeEdit, Riga, Latvia, European Union, 2016.

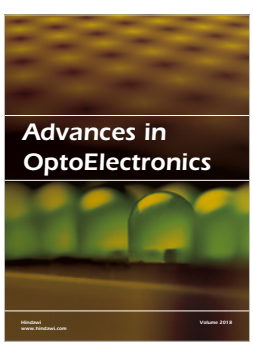
[32] D. Kendall, "Some problems in the theory of queues," *Journal of the Royal Statistical Society*, vol. 13, pp. 151–185, 1951.
[33] G. Bloch, S. Greiner, H. de Meer, and K. Trivedi, *Queueing networks and Markov chains: Modeling and performance evaluation with computer science applications*, John Wiley & Sons, 2006.
[34] J. Little, "A proof of the queueing formula: L=λw," *Operations Research*, vol. 9, no. 3, pp. 383–387, 1961.
[35] R. Muntz, *Poisson departure processes and queueing networks*, IBM Thomas J. Watson Research Center, 1972.
[36] R. Wolff, "Poisson arrivals see time averages," *Operations Research*, vol. 30, no. 2, pp. 223–231, 1982.
[37] M. Alotaibi and A. Nayak, "A distributed approach to improving epc controller performance," in *Proceedings of 86th Vehicular Technology Conference*, pp. 1–6, Toronto, Canada, September 2017.
[38] Ns-3 tutorial, 2013, https://www.nsnam.org/docs/release/3.17/tutorial/singlehtml/index.html.

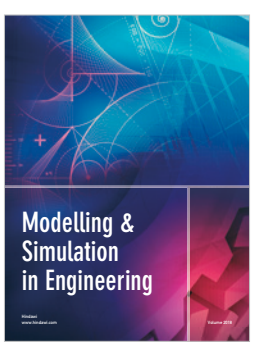


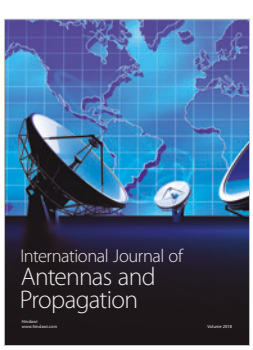


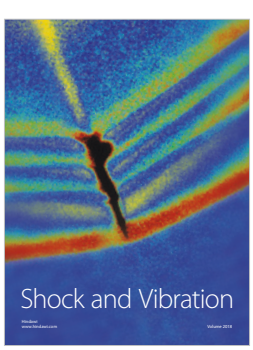


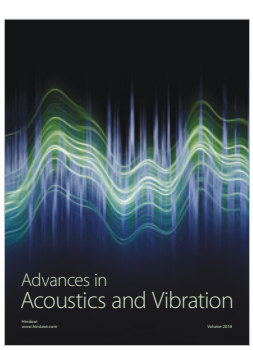












Hindawi

Submit your manuscripts at
www.hindawi.com