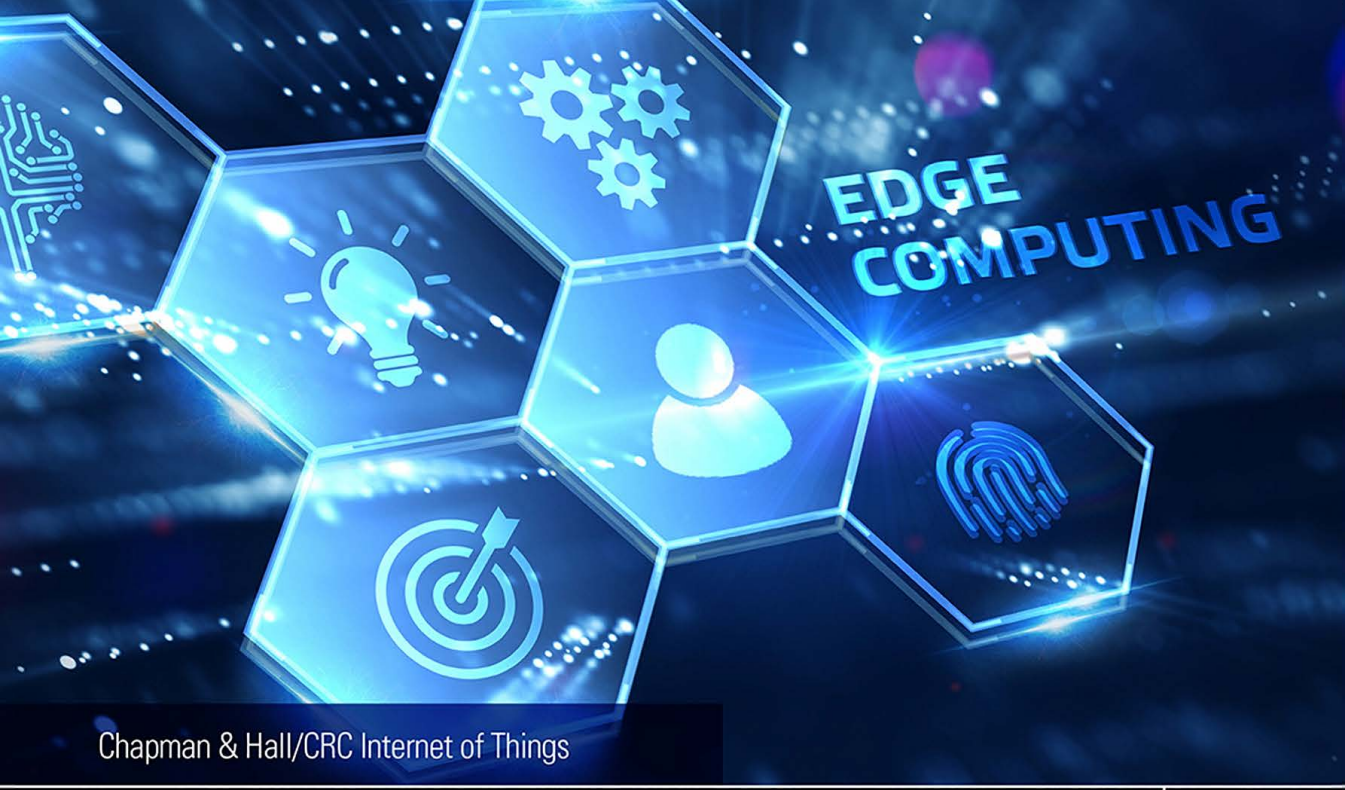

Chapman & Hall/CRC Internet of Things

SDN-SUPPORTED EDGE-CLOUD INTERPLAY FOR NEXT GENERATION INTERNET OF THINGS

Edited by
Kshira Sagar Sahoo
Arun Solanki
Sambit Kumar Mishra
Bibhudatta Sahoo
Anand Nayyar

CRC Press
Taylor & Francis Group
A CHAPMAN & HALL BOOK

Chapman & Hall/CRC Internet of Things: Data-Centric Intelligent Computing, Informatics, and Communication

The interconnected roles of adaptation, machine learning, computational intelligence, and data analytics are becoming increasingly essential in the IoT systems field. IoT-based smart systems generate a large amount of data that cannot be processed by traditional data-processing algorithms and applications. The growing capabilities of intelligent systems depend upon various self-decision-making algorithms in IoT devices. This book series covers the various different computational methods used in the IoT-enabled environment, in particular analytics reasoning, learning methods, artificial intelligence, and sense-making in big data.

This series is aimed at researchers and practitioners working in information technology and computer and data sciences, in particular in intelligent computing paradigms, big data, machine learning, sensor data, and the internet of things. The main aim of the series is to make available a range of books on all aspects of learning, analytics and advanced intelligent systems and related technologies. The series covers the theory, research, development, and applications of learning, computational analytics, data processing, and machine learning algorithms, as embedded in the fields of engineering, computer science, and information technology.

***Series Editors*:**

Souvik Pal
Sister Nivedita University, (Techno India Group), Kolkata, India

Dac-Nhuong Le
Haiphong University, Vietnam

Security of Internet of Things Nodes: Challenges, Attacks, and Countermeasures
Chinmay Chakraborty, Sree Ranjani Rajendran and Muhammad Habib Ur Rehman

Cancer Prediction for Industrial IoT 4.0: A Machine Learning Perspective
Meenu Gupta, Rachna Jain, Arun Solanki and Fadi Al-Turjman

Cloud IoT Systems for Smart Agricultural Engineering
Saravanan Krishnan, J Bruce Ralphin Rose, NR Rajalakshmi, N Narayanan Prasanth

Data Science for Effective Healthcare Systems
Hari Singh, Ravindara Bhatt, Prateek Thakral and Dinesh Chander Verma

Internet of Things and Data Mining for Modern Engineering and Healthcare Applications
Ankan Bhattacharya, Bappadittya Roy, Samarendra Nath Sur, Saurav Mallik and Subhasis Dasgupta

Energy Harvesting: Enabling IoT Transformations
Deepti Agarwal, Kimmi Verma and Shabana Urooj

SDN-Supported Edge-Cloud Interplay for Next Generation Internet of Things
Kshira Sagar Sahoo, Arun Solanki, Sambit Kumar Mishra, Bibhudatta Sahoo and Anand Nayyar

SDN-Supported Edge-Cloud Interplay for Next Generation Internet of Things

Edited by
Kshira Sagar Sahoo
Arun Solanki
Sambit Kumar Mishra
Bibhudatta Sahoo
Anand Nayyar


CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business
A CHAPMAN & HALL BOOK

Front cover image: Den Rise/Shutterstock

First edition published 2023
by CRC Press
6000 Broken Sound Parkway NW, Suite 300, Boca Raton, FL 33487-2742

and by CRC Press
4 Park Square, Milton Park, Abingdon, Oxon, OX14 4RN

CRC Press is an imprint of Taylor & Francis Group, LLC

ISBN: 978-1-032-10149-1 (hbk)
ISBN: 978-1-032-39865-5 (pbk)
ISBN: 978-1-003-21387-1 (ebk)

DOI: 10.1201/9781003213871

Typeset in Palatino
by SPi Technologies India Pvt Ltd (Straive)

6.1 Introduction

Due to rapid advances in internet technology, network terminals have become all but ubiquitous. Legacy network design, on the other hand, has failed to take future communication and internet technology improvements into account. Legacy network infrastructure could not keep up with the rapid development of the internet because it was out of date. Data and control planes are closely connected in conventional network design, which has various constraints. For example, if we wish to modify the configuration of the network, each device must be configured individually throughout the whole network. This is a significant disadvantage. For the same reason, suppliers do not provide developers and members of the community with access to the essential configuration settings of their devices, since doing so might cause networks to malfunction. Protocols are also deeply ingrained in the network equipment's firmware. Due to proprietary hardware and a lack of testing for creative networking solutions, these limitations limit network innovation, increase administrative effort, and drive up network management costs [1–4].

The SDN paradigm [5–8] introduces novelty in computer networks by decoupling the data and control planes. The separation of data and control planes shifts the network complexity from networking devices to the intelligent SDN controllers; thus, the network devices can be programmed through applications running on the controller, while the network is abstracted from applications [9] running on the top of the controller. The network functions according to the applications being executed on the SDN controller. As a result, the complexity of the networks reduces as the logic shifts from the devices to the centralized SDN controller.

However, besides the numerous advantages of SDN, its modeling, evaluation, and testing present several challenges. One of the challenges is the performance evaluation of the controllers [10–12]. The controller plays a prominent role in SDN. Hence, different approaches are used for performance evaluation and comparison of SDN controllers. However, these approaches need proper categorization to enable improvements in SDN research. This chapter discusses, categorizes and analyzes these approaches.

The objectives of the chapter are as follows:

- To explore controller performance evaluation procedures,
- To investigate several schemes for performance evaluation of SDN controllers based upon quantitative and qualitative analysis,
- To evaluate quantitative performance evaluation approaches in Mininet and Cbench,
- To evaluate feature-based comparison of controllers, and
- To evaluate hybrid methods combining quantitative performance evaluation and feature-based analysis.

Organization of the Chapter

The rest of the chapter is organized as: Section 6.2 discuss the basics of the SDN architecture and the important features in SDN controller performance evaluation. Section 6.3 explains various schemes in the literature for controller performance evaluation. Section 6.4

describes the analytical network process for controller selection in SDN. Section 6.5 illustrates the results. Section 6.6 concludes the chapter with future scope.

6.2 SDN Architecture

Generally, the SDN is composed of data, control, and management planes. In this section, we discuss each plane briefly. Figure 6.1 shows these three planes and the interaction among them through southbound (SB) and northbound (NB) APIs.

6.2.1 Data Plane

In SDN, the data plane comprises forwarding devices (known as the infrastructure or the underlying network). The data plane packets are matched and actions are taken in accordance with the forwarding rules described in a flow table.

A flow table is made up of many flow entries. The packet header information is compared to the flow table entries. Each flow entry consists of three fields: a header, an action, and a counter. Table 6.1 illustrates a flow table, with the top row containing header data and the subsequent rows containing flow entries. When a new packet arrives on a switch's ingress port, the matching procedure begins, as shown in Table 6.1. If the packet's destination IP address begins with 192.168.X.X, it is forwarded to port number 1, and counter 102 is updated. Similarly, the third row specifies that if a packet's source and destination port

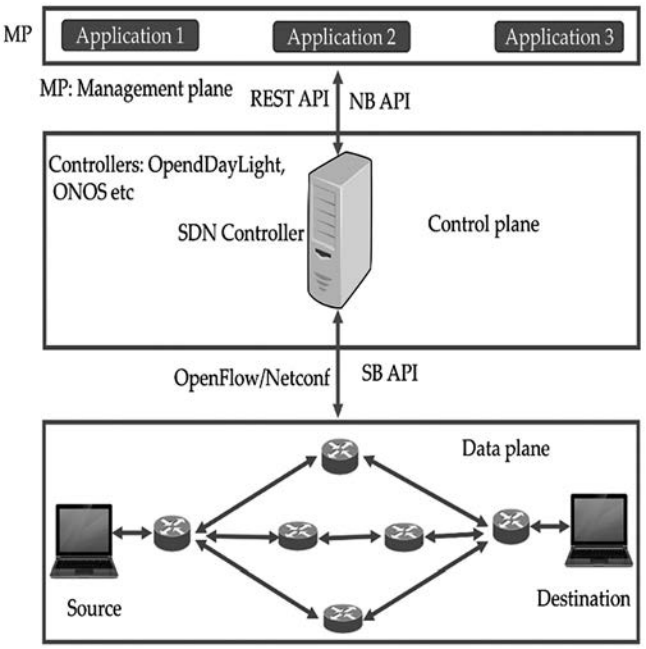

FIGURE 6.1
Three-plane architecture for SDN [9].

TABLE 6.1
An Example of the OpenFlow Table

Source IP	Destination IP	Source Port	Destination Port	Action	Counter
X	192.168.X.X	X	X	Port 1	102
X	X	21	21	Drop	621
192.168.1.X	X	X	X	Port 3	100
X	X	20	80	Port 4	101

numbers are identical, it should be dropped. If no rules for the new packet exist in the flow table, the switch sends a packet-in message to the controller, which returns the destination to the forwarding device (packet-out message) and updates the flow rules in the flow table accordingly. This is in contrast to typical networks, where routing decisions are made between closely coupled devices.

6.2.1.1 Southbound (SB) APIs

The SB-API interfaces the data and control planes. Various protocols, such as OpenFlow [13] and Netconf [14], are available for interfacing the two planes. OpenFlow is the most widely used protocol in the SDN community. OpenFlow provides a secure communication channel between the controller and the switches. The white paper [15] discusses the benefits and programmability of OpenFlow for forwarding device programming. The OpenFlow idea originated at Stanford University, and the Open Networking Foundation (ONF) [16] consortium currently manages the OpenFlow standardization process.

6.2.1.2 Northbound (NB) API

The NB-API interfaces the control and management planes. By employing the representational state transfer (REST) API, the controller acts as a bridge between the forwarding devices and the management plane. Similarly, this API is used to retrieve operational statistics (for example, regarding flow entries) from the data plane. Applications operating in the management plane connect with the controller using this API, and the data plane executes the relevant actions. These actions determine the behavior of the management plane application. For instance, a firewall program establishes rules that regulate the admission and egress of packets across the network. As a result, the data plane devices will forward or restrict traffic according to the application's policies. Similarly, a load-balancing program will manage traffic by detecting congestion in various network channels.

6.2.2 Management Plane

The management plane is located above the control plane, and it is where several programs may be run to accomplish the diverse activities required for an effective network. The data plane makes use of the management plane's flexibility and programmability, as well as the abstractions offered by the control plane. For instance, network monitoring may be accomplished comprehensively by the development and deployment of a snipping program. Other security programs may be used to identify and mitigate distributed denial of service (DDoS) threats in a similar fashion [17].

6.2.3 Control Plane

The control plane constitutes the most prominent part of the SDN infrastructure because the entire network is dependent upon it. It is implemented through the SDN controller. Several SDN controllers are available, the most important of which are NOX [18], POX [19], Ryu [20], Floodlight [21], TREMA [22], ODL [23] and ONOS [24]. Each controller has a number of features as discussed below.

(1) *OpenFlow:* OpenFlow is also known as the SB-API which directs the flow requests forwarded by the switches to the controller and vice versa. The controller sends flow response (PACKET_OUT) messages in response to flow request (PACKET_IN) messages. Thus, OpenFlow [13] manages the exchange of messages between these two planes. The effect on SDN controller delay of the exchange of request and response packets is discussed in [25]. However, different versions of OpenFlow are supported by each controller, for example, v1.3 has support for load balancing that facilitates performance improvement during high traffic generation by network devices.

(2) *GUI:* A graphical user interface (GUI) for a controller helps in obtaining statistical information about the underlying forwarding devices, the configuration of switches' flow entries and application deployment. The GUI is considered an important feature in controller selection for SDN and in qualitative and experimental analysis of the controllers [26]. Generally, the SDN controllers are configured through a command-line interface (CLI) or a GUI, or both. The statistics provided by the GUI of the controller consist of information regarding hosts and forwarding devices, flow entries, flow tables and creation of the SDN topologies [27]. Viewing these statistics through the GUI is user friendly and they can be easily examined for analysis. Likewise, flow entries can be inserted to the switches via the GUI, though its features affect performance because the execution of the GUI is slower than the CLI. Further, the GUI support with SDN controllers can be divided into two categories: web-based supported by Java, and Python based. The execution speed of controllers coded in Python is slow because of less support for fast memory access and multithreading.

(3) *Northbound REST API:* In the management plane, communication between applications and controller is through a REST API. Similarly, operational statistics for switches and topologies are collected via this API. The controller uses the REST API and acts as a bridge between the management plane and the data plane. This is a significant aspect of controller selection since it communicates with the controller directly. Rapid response from the API reduces latency and improves throughput. Hence, the REST API is important for SDN performance [28, 29] and controller selection.

(4) *Clustering:* Native support for clustering tends to improve the scalability, stability, and performance of the controller. Controllers that support clustering have improved performance with respect to latency [30]. Similarly, decreased latency is observed with increased switches. The same effect is observed during high traffic generation through the clustering of controllers.

(5) *Quantum API:* Users leverage the services provided by the cloud by calling them remotely over the internet. Thus, there is a competition between cloud service providers (CSPs) and network service providers (NSPs). Huang et al. [31] proposed an economic model that distinguishes the competition between CSP and NSP.

Similarly, the research conducted by Huang et al. in [32] described a way to provide E2E performance using a cloud services configuration model. With support for the Quantum API, SDN controllers can take advantage of cloud computing. Controllers that natively support this API can use the Quantum API to leverage the cloud in the realm of high-performance computing as well as OpenStack networking. Controllers with quantum support feature parallelism and fast memory access. Consequently, performance improves with SDN scalability, i.e., when the number of switches increases. This API was incorporated into a research study describing controller selection [28].

(6) *Synchronization:* This shows how effectively a controller stores and responds to information about the network, which affects the time taken to discover the topology. This is very important for performance monitoring in SDN [33]; controllers which take a short time to discover the topology improve SDN performance.

(7) *Productivity:* This relates to the ease of development of applications and to the programming language of the controller. Using a controller coded in Python makes it easy to develop your application, but without the platform support and multithreading, it is slow. Productivity is therefore, inversely related to controller performance [34]. Well-coded Python is easier to develop applications and more productive, but lower productivity means better Java support. This is due to the fast memory access, cross-platform support, multithreading, and inter-process communication (IPC) features of Java-supported controllers which lead to high performance.

(8) *Partnership support:* Many local and international organizations support SDN controllers. As a result, IT organizations not only consider technical strengths when choosing a controller but also key aspects such as the financial resources as well as the sponsors associated with a controller. This function plays a vital role in controller selection [34].

(9) *Platform support:* This concerns the compatibility of SDN controllers with different operating systems, such as Linux, Mac, or Windows. Executing compatibility across various platforms means the controller has support for multithreading, flexible management of memory and fast memory access affecting controller performance. Because controllers can create clusters on different platforms, running on other platforms improves the efficiency of clustering. This reduces latency and improves QoS during normal and high traffic loads [30].

(10) *Modularity:* The ability to make the sub-routines of the main program is called modularity. Modular support helps the controller in dealing with large-scale systems. If controller modularity is high, it means the sub-modules have the capability to run in parallel, which speeds up execution and hence reduces response time. It helps to improve performance, especially when scalability increases.

6.3 Categorization of SDN Controller Selection Approaches

Several methods of SDN controller selection are described in the literature. These methods can be divided into three main categories: comparing controllers by performance, features, and hybrid. The hybrid approach combines feature results with performance-based comparisons to select the best controller. These methods are described below.

6.3.1 Performance Analysis Controller Selection

The studies presented in [35], [36, 37] and [38, 39] compare the controllers through performance. The performance-based approach only considers the performance and neglects features of the SDN controller. This approach only considers the performance of a typical topology made in Mininet or through creating virtual hosts and switches in Cbench. Hence, realistic scenarios of the real internet are not taken into consideration in the experiment.

6.3.2 Feature-Based Controller Selection

The research carried out in [40–43] compares only the supporting features of SDN controllers. Examples of support features are platform, OpenFlow, REST API, and clustering. Another disadvantage of this approach is that it provides only a theoretical analysis of the feature set that the controller provides. This selection method will result in a cognitive overload. Thus, the optimal decision cannot be made due to the 7 ± 2 problem, or the limitations of the human memory for processing information, also known as Miler's law [44].

6.3.3 Hybrid Methods of Controller Selection

A comparison of four SDN controllers using a hybrid approach was highlighted by Bispo et al. [26]. The authors chose two controllers based on heuristic decisions in the table of nine controller functions and ran Cbench in throughput and latency mode to evaluate the performance of those controllers. The study did not provide an explicit ranking of these controllers as study only analyzed the feature table, making accurate choices impossible. Second, authors did not consider performance comparisons in a real internet topology. The study by Salman et al. [34] compared the six SDN controllers Opendaylight (ODL), Beacon [45], Nox, Maestro [46], Libfluid Raw [47] and Ryu with respect to an increase in the number of switches and threads in the latency and throughput modes. Another study by Mamushiane [48] compared the performance of four controllers consisting of Ryu, ODL, Open Network Operating System (ONOS), and Floodlight. In this study, the authors used Cbench for measuring latency and throughput. Anderson [33] compared five SDN controllers – Trema [22], Ryu, ODL, ONOS and Floodlight – for wireless networks using qualitative and quantitative studies. A qualitative study of two features of these controllers – clustering support and state handling – was carried out first. Thus, in case of switch or controller failure, information about the state was tabulated for the five controllers to see how each controller collects and stores network state information and the status of this information, i.e., whether the controller reloads this information from a previously saved state or relearns the network state. Likewise, the clustering information was tabulated to see if these controllers support clustering and how other controllers share information among the clusters of controllers. In the study, two controllers were selected based on these two features that met the constraints of the aerial networks. The performance of the two controllers was then evaluated through emulated experimental scenarios in Mininet. Nevertheless, the selection process for controllers is based on heuristic decisions, and as the number of controllers and features expands, cognitive overload can occur, leading to disobedience of Miller's law.

6.3.4 Multi-Criteria Decision-Making (MCDM) Methods for Controller Selection in SDN

MCDM is a mathematical decision-making technique used for the selection of alternatives based on certain criteria elements [49]. It has been extensively used in many fields, such as for strategy selection in software development [50], for natural resources management

[51] and for selection of a network among heterogeneous networks [52], etc. Various approaches are in use for the selection process, which depends on several criteria elements to achieve the desired objective, for example, TOPSIS, analytical hierarchy process (AHP) and ANP, etc. An AHP method of selecting an SDN controller was proposed by Khondoker et al. [28]. The research sought to select a controller from ten candidates considering only their features, but did not perform any experimental evaluation of the controllers, nor did it provide the mathematical details of the approach.

A hybrid scheme based on AHP which utilizes both the features and performance evaluation for controller selection is described in [29]. Based on the feature set selection, the three highest-ranked controllers were evaluated using Cbench, but the performance in real internet topologies was not evaluated. AHP does not take into account the feedback from the alternatives, and it only considers the criteria weights for selection of alternatives. Another problem with AHP is that it does not consider the dependency of the criteria elements, making precise and accurate selection impossible. In [53], ANP was used to model risk factors in large projects using risk indices. Similarly, Nazir et al. [54] used quality criteria to select software components. ANP has been used in wireless sensors for optimal cluster head selection [55]. Therefore, it is assumed that the ANP approach may be used to analyze systems with complex behaviors and structures. Due to the complexity of the systems, the dependencies between them have increased. Therefore, research on interdependent network systems is important [56]. ANP is an established tool for the decision-making process where dependency exists among criteria elements.

6.4 Analytical Network Process-Based Controller Selection

The ANP MCDM problem is formulated by first defining the aim or objective, then defining the parameters for the criteria or sub-criteria, and then considering the assessment options. After listing the controller's characteristics, ANP is applied. Figure 6.2 illustrates the technique in detail. The purpose of this research is to determine the optimal SDN controller based on the ten characteristics listed in Table 6.2. Equations (6.1) and (6.2) indicate the criteria and alternatives. F denotes the available features of the various SDN controllers, while C denotes the alternatives. It is necessary to create a network model that represents the criteria and options, as well as their connection. Each possibility was analyzed in terms of each criterion and vice versa using the network model.

$$F = \left(F_1, F_2, F_3, \ldots, F_N\right) \tag{6.1}$$

$$C = \left(C_1, C_2, C_3, \ldots, C_N\right) \tag{6.2}$$

The ten critical characteristics that should be examined while selecting SDN controllers are described below. We presume that each of these characteristics is necessary for the controller selection process. However, since controllers are constantly changing, we use the most up-to-date information on these aspects from controller documentation and research reported in [25, 26]. These critical characteristics are taken into account in the ANP-based optimal controller selection procedure. The significance of a feature in a controller is

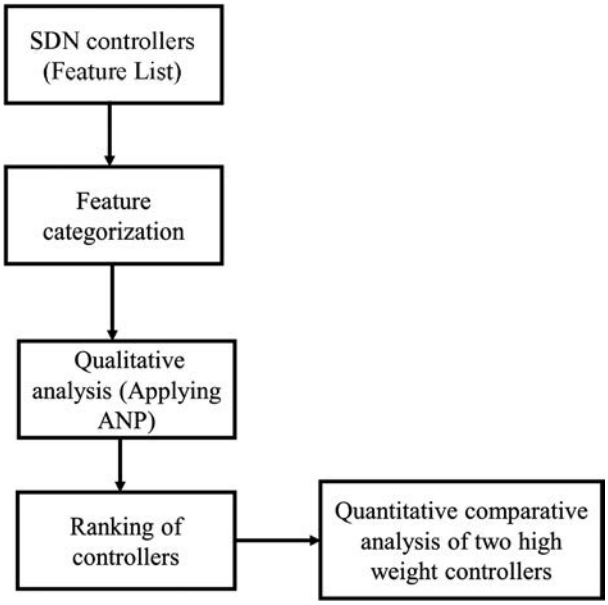

FIGURE 6.2
The procedure for controller selection leveraging ANP.

TABLE 6.2
Features Categorization for SDN Controller Selection

	Criteria/Features									
Alternatives	**F_1**	**F_2**	**F_3**	**F_4**	**F_5**	**F_6**	**F_7**	**F_8**	**F_9**	**F_{10}**
C_1	M	VH	Yes	Yes	No	M	M	L	L	M
C_2	H	H	Yes	Yes	Yes	M	M	VH	H	H
C_3	H	H	Yes	Yes	Yes	L	M	H	H	H
C_4	L	M	No	No	No	L	H	L	H	L
C_5	VH	L	No	Yes	No	H	H	M	L	M
C_6	L	L	No	Yes	No	L	H	L	L	M

determined by classifying these characteristics. A controller may handle two distinct sorts of features: ordinal and regular. Ordinal features have an intrinsic ordering, but regular categorical features do not. The classification of the feature set shows the extent to which each feature is supported by each controller. For example, C_4 and C_6 support only OpenFlow v1.0, and hence fall into the low category (L) for this feature (F1). C_1 support is medium (M), C_2 and C_3 support v1.0,1.1,1.3, respectively, and are therefore classified as high (H), whereas C_5 supports higher versions of OpenFlow, namely 1.5, and is thus classified as very high (VH). F_2 denotes a controller's GUI. C_1 supports a Java-based web interface and is quicker to run due to the presence of graphical tools for application and data plane administration.

As a result, F_2 is classified as very high due to its usage of Java multithreading. Similarly, C_2 and C_3 feature a Java-based interface and allow for the configuration of QoS parameters for data plane devices. They offer more application administration and topology setup options, which results in a slower GUI than C1. As a result, they are classified as high. C_4 has a Python-based interface and has a quicker execution speed than C_5 and C_6 due to its preparatory functions; hence, it is classified as medium. C_5 and C_6 offer solely Python-based interfaces; nevertheless, they perform poorly owing to the increased number of functions to maintain the data plane and management plane, as well as the absence of multithreading.

F_3, F_4, and F_5 are regular categorical traits, i.e., characteristics that cannot be subdivided further. A controller, for example, may or may not support REST APIs, open stack networking, or clustering. As a result, these features (F_3, F_4, and F_5) lack intrinsic ordering. In Table 6.2, these characteristics are denoted by Yes or No. As with C_4, C_5 and C_6 lack built-in support for REST APIs; hence, they get a No in the equivalent F_3 column of Table 6.2. Similarly, C_1, C_2, and C_3 provide built-in support for the REST API; hence, the F_3 column is marked Yes. F_4 demonstrates the Quantum API. Because C_1, C_2, C_3, C_5, and C_6 have an inbuilt support for the Quantum API, a value of Yes is shown in column F_4 for them. C_4 does not support the Quantum API; hence, No appears in F_4 for this controller. The clustering characteristic is denoted by F_5. Because the controllers C_1, C_4, C_5, and C_6 lack built-in capability for clustering, a No is entered in the F_5 column of Table 6.2. In comparison, C_2 and C_3 support clustering and are therefore denoted by Yes.

F_6 denotes the synchronization characteristic that has an effect on the data plane's topology discovery and response. C_1, C_5, and C_6 have a medium amount of interaction, which means that their interactions are rather sluggish. C_2 and C_3 are considered high level due to their rapid discovery of the underlying topology, whilst C_4 is considered low level due to the slowest interaction between the data and control planes. F_7 indicates a controller's production level. Productivity is connected to the simplicity with which an application may be developed and is dependent on the programming language used to write the controller. While developing applications using Python-coded controllers is straightforward, their lack of platform support, memory management, and multithreading make them sluggish. C_1, C_2, and C_3 have a moderate degree of productivity, whilst C_4, C_5, and C_6 have a high amount.

F_8 indicates the presence of support from many manufacturers. C_2 is backed by Cisco, NEC, IBM, and the Linux Foundation, which has a membership of over 40 corporations; as a result, it is rated very high. C_3 is backed by SK Telecom (South Korean telecommunications), Cisco, and NEC, and is therefore classified as high. C_1, C_4, and C_6 are supported by Big Switch Networks, Nicira, and NEC, respectively, and are therefore classified as low in terms of support. Although it is not directly connected to performance, effective vendor support ultimately results in performance improvement. F_9 is the platform support key. C_2, C_3, and C_4 all support three platforms, namely Linux, Mac, and Windows, and are therefore classified as high end. However, since C_1, C_5, and C_6 are supported on just one platform, namely Linux, they are accorded a low grade. Cross-platform compatibility allows multithreading and clustering, resulting in an increased quality of service. F_{10} indicates that modularity is supported. C_1, C_5, and C_6 have a moderate amount of modularity, but C_2 and C_3 have a high level of modularity. This is because C_2 and C_3 controllers may invoke sub-modules from the main function, resulting in parallel processing and therefore increasing performance. Categorization of features is performed as a pre-processing step prior to creating the comparison matrix.

6.4.1 Pairwise Comparison Matrix for Criteria and Alternatives

The pairwise comparison matrix is drawn up according to the 9-point scale proposed by Saaty [57] as shown in Table 6.3. It shows the relative importance of different components (criteria or alternatives) regarding an element. The same matrix is also employed to extrapolate the effect of the components on the objective using the 9-point scale as shown in Table 6.3. The values in the matrix are assigned to the criteria as well as alternatives representing personal judgments. For example, the importance of C_1 compared with C_2, C_3, C_4, and C_5 with respect to the F_1 component is assigned a value from the scale table. The value of $a_{(i,j)}$ represents the relative significance of a component corresponding to the i^{th} row and row j^{th} column. The value of $a_{(i,j)} = 1$ in the pairwise comparison matrix shows the equal importance of the component corresponding to the i^{th} row and j^{th} column. The diagonal components correspond to the comparison of the same components; therefore, their values are 1. The values below the diagonal are the reciprocal of the values above the diagonal. The value of, $a_{(5,1)} = 6$ shows that component in the 5th row is significantly to remarkably more important than the component in the 1st column. The value of $a_{(1,5)} = \frac{1}{a_{(5,1)}} = \frac{1}{6}$ is the reciprocal of $a_{(5,1)}$, denoting that a component in the 1st row is significant to remarkably less important than a component in the 5th column. The values are incorporated prudently for all the components in the pairwise comparison matrix.

6.4.2 Pairwise Comparison Matrix for Criteria and Alternatives

Alternatives are pairwise compared to each criteria component. The general form of the pairwise comparison matrix is denoted in the matrix (6.3). The rows and columns of the matrix are represented as M_1 to M_n and N_1 to N_n. First, the alternatives are pairwise compared with respect to the F_1 criterion. The values have been incorporated in (6.3) based using the 9-point scale defined in Table 6.3. The nonreciprocal and reciprocal values indicate the relative importance of the row and column components, respectively. First, the comparison of C_1 is made with C_2, C_3, C_4, C_5, and C_6 considering F_1 criterion. C_1 is of the same importance as itself therefore $a_{(1,1)}$=1. Then C_2 and C_3 are moderately more important

TABLE 6.3

Values for Relative Importance of One Feature/Controller Over Another

Scale	Description
1	Equally important
2	Equally to moderately more important
3	Moderately more important
4	Moderately to significantly more important
5	Significantly more important
6	Significantly to remarkably more important
7	Remarkably more important
8	Remarkably to excessively more important
9	Excessively more important

than C_1. i.e. , $a_{(1,2)} = a_{(1,3)} = \frac{1}{3}$. C_1 moderately more important than C_4 and C_6, e.g., $a_{(1,6)} = 3$ shows that the alternative in this row *(C_1)* is moderately more important than the alternative in the corresponding column *(C_6)*. $a_{(1,5)} = \frac{1}{6}$ shows that C_5 is significant to remarkably more important than C_1. Similarly, the values are filled for C_2–C_6.

$$\begin{bmatrix} & N_1 & N_2 & N_3 & \cdots & N_n \\ M_1 & 1 & a_{(1,2)} & a_{(1,3)} & \cdots & a_{(1,n)} \\ M_2 & \frac{1}{a_{(1,2)}} & 1 & a_{(2,3)} & \cdots & a_{(2,n)} \\ M_3 & \frac{1}{a_{(1,3)}} & \frac{1}{a_{(2,3)}} & 1 & \cdots & a_{(3,n)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ M_n & \frac{1}{a_{(1,n)}} & \frac{1}{a_{(2,n)}} & \frac{1}{a_{(3,n)}} & \cdots & 1 \end{bmatrix} \tag{6.3}$$

Then, matrix (6.4) is used for summing up and each value is divided by the sum of the total values of the column. The next step is to find the eigenvector from the normalized matrix. The eigenvector shows the priority of these features F_1.

$$\begin{bmatrix} \frac{a_{(1,1)}}{\sum_{i=1}^{n} a_{(i,1)}} & \cdots & \frac{a_{(1,n)}}{\sum_{i=1}^{n} a_{(i,n)}} \\ \vdots & \ddots & \vdots \\ \frac{a_{(n,1)}}{\sum_{i=1}^{n} a_{(i,1)}} & \cdots & \frac{a_{(n,n)}}{\sum_{i=1}^{n} a_{(i,n)}} \end{bmatrix} \tag{6.4}$$

The eigenvector X is obtained from the normalized matrix (6.4) according to Equation (6.5).

$$X_i = \frac{1}{n} \sum_{j=1}^{n} a_{(i,j)}, \text{where i} = 1, 2, 3, \ldots, n \tag{6.5}$$

The result from Equation (6.5) is considered as the eigenvector X_1. To verify whether the judgments made while making the pairwise matrix are consistent, the next step is to find the *CI* and *CR* values. However, before making the consistency analysis, the consistency measure *(CM)* vector is to be calculated.

Consistency Measure: The *CM* vector is a prerequisite for the calculation of *CI* and *CR*. The consistency measure is calculated according to Equation (6.6). M_j denotes the row of the comparison matrix (6.3). X and x_i represents the eigenvector and the corresponding element of the eigenvector as shown in the matrix (6.5). The M_j and X are multiplied and

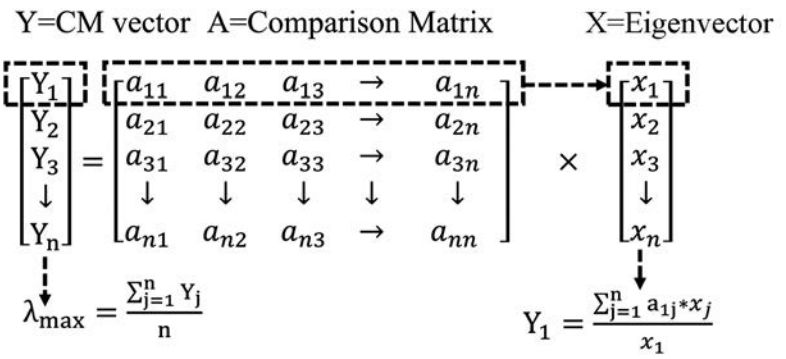

FIGURE 6.3
The measurement of the consistency measure.

then divided by the component in the eigenvector corresponding to M_j. The procedure to find the *CM* is shown in Figure 6.3. The *CM* vector is averaged for computing λ_{max}.

$$Y_j = \frac{M_j * X}{x_i}, \text{where } j = 1, 2, 3, \ldots, n \tag{6.6}$$

$$\lambda_{\max} = \frac{1}{n} \sum_{j=1}^{n} Y_j \tag{6.7}$$

Consistency Index: CI denotes the deviation or the inconsistency [54] of the pairwise comparison matrix for an element. The *CI* of the pairwise comparison matrix for the F_1 criterion is calculated using Equation (6.8) by putting the value of λ_{max}. The value of λ_{max} = 6.07 and n = 6 are put in Equation (6.8).

$$CI = \frac{(\lambda_{\max} - n)}{(n-1)} \tag{6.8}$$

In Equation (6.8), n represents the criterion number for controller selection in the comparison matrix. Here, six alternatives are considered; therefore, n is equal to 6. The resultant value for *CI* = 0.01, according to Equation (6.8).

Consistency Ratio: The reliability of the pairwise comparison matrix is verified by calculating the *CR* value. The *CR* is calculated according to Equation (6.9). In Equation (6.9) the ratio index *(RI)* denotes the index ratio. The value of *RI* = 1.24 is derived from Table 6.4, based on the order of the matrix. If the rank of the matrix is three (the actual number of alternatives being compared), then a value corresponding to three is selected for *RI*. In this case, the number of criteria under consideration is 6. Therefore, a value corresponding to 6 will be inserted from Table 6.4. The *CR* is derived by putting *CI* value from Equation (6.8) into Equation (6.9).

$$CR = \frac{CI}{RI} \tag{6.9}$$

TABLE 6.4
Ratio Index for Different Number of Criteria

Criteria	1	2	3	4	5	6	7	8	9	10
Ratio Index	0.00	0.00	0.58	0.90	1.12	1.24	1.32	1.41	1.45	1.49

The *CR* value is 0.09. A *CR* of 0.1 or less is accepted for the inconsistent judgments of the comparison matrix; otherwise, the inconsistency is high and pairwise judgments must be made again to satisfy the condition, i.e., $CR \leq 0.1$. The alternatives are pairwise compared to the remaining criteria, i.e., F_2, F_3, F_4, F_5, F_6, F_7, F_8, F_9, and F_{10}. The *CI* and *CR* values are computed using the same process for each of these matrices. The *CR* value is shown in each matrix.

Likewise, the eigenvectors corresponding to X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8, X_9, and X_{10} are computed corresponding to features F_1, F_2, F_3, F_4, F_5, F_6, F_7, F_8, F_9, and F_{10} respectively along with their *CR* values. X_1 represents the eigenvector corresponding to the F_1 criterion. Similarly, X_2 represents the eigenvector for the F_2 criterion, X_3 for F_3 and so on. The *CR* value for calculating each eigenvector is verified to be less than 0.1.

6.4.3 Pairwise Comparison for Criteria with Respect to Controllers

The ten features F_1, F_2, F_3, …, F_{10} of the criteria are pairwise compared for all alternatives C_1, C_2, C_3, C_4, C_5, and C_6. The corresponding eigenvectors for these alternatives are calculated, i.e., X_{11}, X_{12}, X_{13}, X_{14}, X_{15}, and X_{16}. The eigenvectors for alternatives should be calculated using similar calculations as we have done for criteria elements. The *CM, CI*, λ_{max} and the *CR* values for each matrix were calculated. The *CR* value for each eigenvector was checked and verified to be less than 0.1 for maintaining consistency among judgments.

6.4.4 Weighted Super-Matrix

The eigenvectors are computed (which indicates the weight of each criterion in relation to each option and vice versa). We then combine them to form an unweighted super-matrix. Additionally, the unweighted super-matrix is modified to be column stochastic, so that the total of each column equals one. This operation converts the matrix to a super-matrix with weights. The weighted super-matrix illustrates the comparison of criterion alternatives and vice versa. The unweighted super-matrix is identical to the weighted super-matrix; however, the weighted super-matrix is column stochastic. X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8, X_9, and X_{10} are the eigenvectors respectively for features (i.e., F_1, F_2, F_3, F_4, F_5, F_6 F_7, F_8, F_9, F_{10} which denote the priority values of the features) for each controller. Similarly, X_{11}, X_{12}, X_{13}, X_{14}, X_{15}, X_{16}, the eigenvectors corresponding to C_1, C_2, C_3, C_4, C_5, and C_6, show the priority of the alternatives (controllers) regarding each feature. Finally, we compute the limit super-matrix in order to derive the final weights for the alternatives in the ANP model given in the next paragraph.

6.4.5 Super-Matrix (Limit) Computation

The weighted super-matrix is processed by increasing its power until it converges to a stable matrix. The stable matrix is also referred to as the limit super-matrix. The limit matrix indicates the relative importance of the alternatives and criteria, i.e., the final prioritized values. As a consequence, the limit matrix comprises the final weights assigned to each

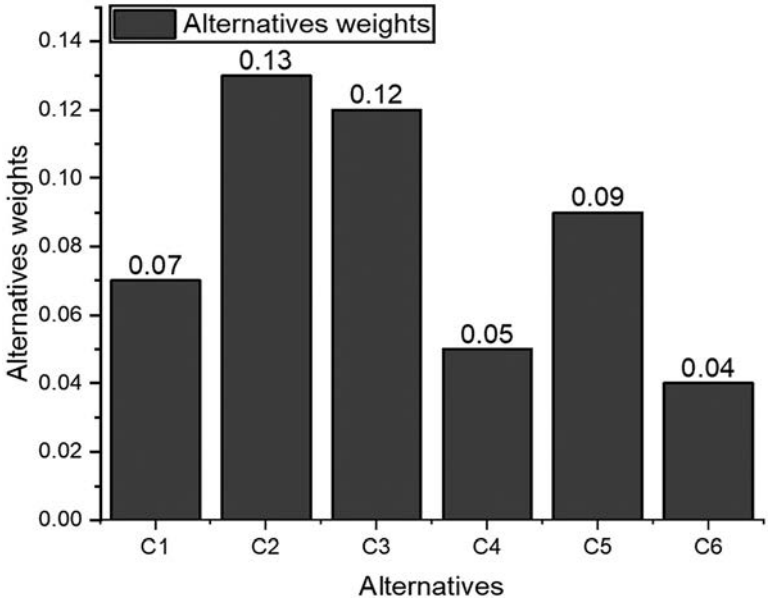

FIGURE 6.4
Priorities of controllers computed by the limit super-matrix.

element in the criterion and alternative clusters. It is computed using a weighted super-matrix in which values are raised to the power of 2^k to produce the identical value for each row, where k is any random integer. The limit super-matrix aggregates all matrices' pairwise comparisons. Additionally, it demonstrates the indirect interaction between the components. The results of the limit super-matrix are shown in Figure 6.4, where heavy weight represents the standing alternative. Figure 6.4 shows the final stable weights of all options. As C_2 has the largest weights, it is the best suited controller. According to their final weights determined from the limit super-matrix, the following controllers are suitable: C_3, C_5, C_1, C_4, and C_6. According to the findings, C_2 has a high weight value, and hence the suggested SDN controller with ANP model corresponds to it. Additionally, the suggested controller's performance is tested in the next part via tests, and then compared to the controller proposed using the AHP model. The following section covers the findings of experimental simulations for both AHP and ANP controllers.

6.5 Results and Discussion

The performance of the C_2 controller (ODL) calculated using the suggested technique, i.e., ANP, is to be analyzed. Additionally, performance is compared to that of the AHP-generated Ryu controller. The Mininet Python API was used to simulate the network topologies calculated using the proposed and AHP approaches. This network simulator has been frequently used to prototype experiments based on SDN. Mininet 2.3.0d1 and OpenvSwitch (OVS) 2.5.4 were installed in Ubuntu 16.04 LTS. Additionally, the Xming server was launched in order to create and visualize traffic between the source and destination servers.

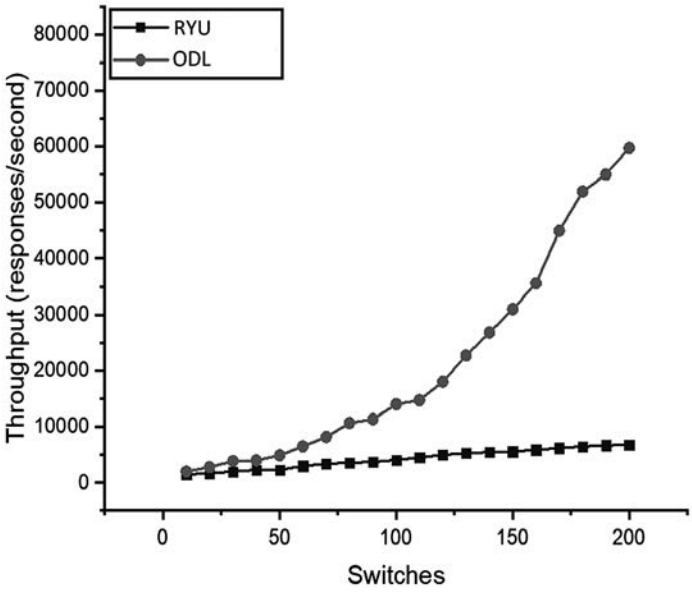

FIGURE 6.5
The throughput evaluation with scalability of the switches in SDN.

6.5.1 Throughput Evaluation

Throughput is determined by sending PACKET-IN packets to the controller and computing PACKET-OUT (responses/second) packets using Cbench. The number of MACs per switch is limited to 2000, the switches are changed between 100 and 200, and each test is repeated ten times. The findings indicate that the ANP controller's throughput does not diminish and that it starts quickly, as seen in Figure 6.5.

6.5.2 Utilization of CPU

To determine the CPU consumption, we utilized a program called sysbench and evaluated both ODL and Ryu controllers. The findings for CPU use at 20-second intervals are shown in Figure 6.6. The graph demonstrates that at peak controller use, utilization does not exceed 45% for a controller with ANP and 30% for a controller with AHP. However, with typical traffic, the controller suggested with AHP achieves a maximum utilization of 19% while the controller proposed with ANP achieves a maximum utilization of 26%.

6.6 Conclusion and Future Scope

The primary objective of this study was to examine numerous strategies for selecting controllers for SDN. The selection of a controller is influenced by a number of factors, including platform support, northbound and southbound interfaces, productivity, and modularity. As a result, we defined it as an MCDM issue. Additionally, the ANP MCDM was used to resolve this issue. To begin, we determined the characteristics that affect performance,

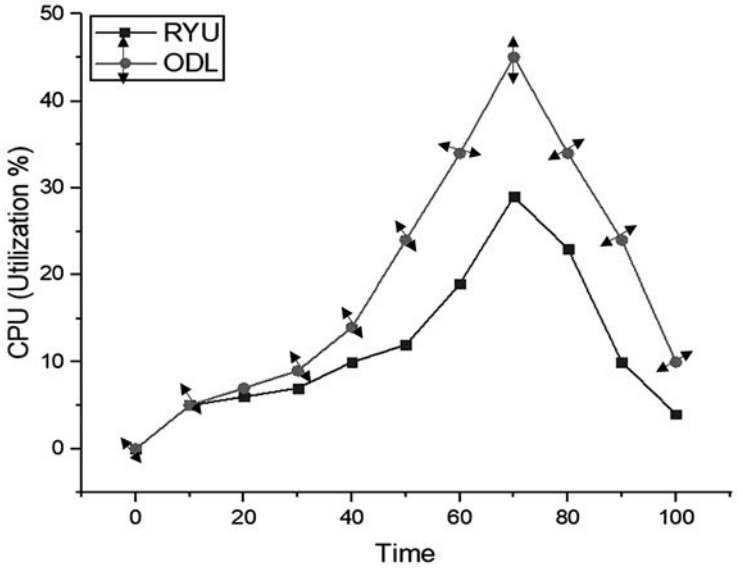

FIGURE 6.6
Percentage of CPU utilization.

namely the criterion parameters and the alternatives (controllers). Then, pairwise comparisons between each characteristic in the criteria cluster and each alternative in the controller's cluster were performed, and vice versa. We obtain priority vectors (eigenvectors) from these comparisons, which are then put in an unweighted super-matrix that has been column stochastically transformed to produce a weighted super-matrix. The final matrix, which indicates how the options and criteria are prioritized, is a limit super-matrix. The limit super-matrix findings indicate that the C_2 controller (ODL) has the best feature set among the SDN controllers investigated in this research. Thus, a high-priority or high-weight controller from the limit super-matrix was chosen for further experimental study.

To validate the performance of the feature-based optimal controller, i.e., ODL, we compared it to an AHP-based controller for the same feature set. We compared the two controllers experimentally by examining several QoS indicators, such as CPU use and throughput. We confirmed the experimental results using Mininet, demonstrating that ODL (ANP) beats Ryu (AHP). We generated a controller with a higher priority weight for supporting features than previous controllers using the suggested technique, and the experimental study confirmed in Mininet demonstrated an increase in performance with the ANP controller.

In future, we tend to investigate other MCDM methods for performance benchmarking of SDN controllers. Hence, a comparison will be made with other MCDM schemes to rank the controllers.

References

[1] J. Ali, and B. H. Roh, "An effective hierarchical control plane for software-defined networks leveraging TOPSIS for end-to-end QoS class-mapping," *IEEE Access*, vol. 11, no. 8, pp. 88990–9006, 2020 May.

[2] K. S. Sahoo, M. Tiwary, B. Sahoo, B. K. Mishra, S. RamaSubbaReddy, and A. K. Luhach, "RTSM: response time optimisation during switch migration in software-defined wide area network," *IET Wireless Sensor Systems*, vol. 10, no. 3, pp. 105–111, 2020.
[3] K. S. Sahoo, D. Puthal, M. S. Obaidat, A. Sarkar, S. K. Mishra, and B. Sahoo, "On the placement of controllers in software-defined-WAN using meta-heuristic approach." *Journal of Systems and Software*, vol. 145, pp. 180–194, 2018.
[4] S. Nithya, et al., "SDCF: A software-defined cyber foraging framework for cloudlet environment," *IEEE Transactions on Network and Service Management*, vol. 17, no. 4 pp. 2423–2435, 2020.
[5] 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.
[6] H. Farhady, H. Y. Lee, and A. Nakao, "Software-defined networking: a survey," *Computer Networks*, vol. 81, no. C, pp. 79–95, 2015.
[7] N. Mc J. H. Cox et al., "Advancing software-defined networks: A survey," *IEEE Access*, vol. 5, pp. 25487–25526, 2017.
[8] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.
[9] J. Ali, G. M. Lee, B. H. Roh, D. K. Ryu, and G. Park, "Software-defined networking approaches for link failure recovery: A survey," *Sustainability*, vol. 12, no. 10, p. 4255, 2020.
[10] J. Ali, B. Roh, and S. Lee, "QoS improvement with an optimum controller selection for software-defined networks," *PLoS One*, vol. 14, no. 5, p. e0217631, 2019.
[11] J. Ali, and B. H. Roh, "Quality of service improvement with optimal software-defined networking controller and control plane clustering," *Computers, Materials and Continua*, vol. 67, no. 1, pp. 849–75, 2021 Jan 1.
[12] J. Ali, S. Lee, and B. H. Roh, "Performance analysis of POX and Ryu with different SDN topologies," in *Proceedings of the 2018 International Conference on Information Science and System* (pp. 244–249), 2018 Apr 27.
[13] S. J. Vaughannichols, "OpenFlow: The next generation of the network," *Computer*, vol. 44, no. 8, pp. 13–15, 2011.
[14] J. Crawshaw, "NETCONF/YANG: What's holding back adoption & how to accelarate it," Heavy Reading Reports, 2017.
[15] A. Lara, A. Kolasani, and B. Ramamurthy, "Network innovation using OpenFlow: A survey," *IEEE Communications Surveys and Tutorials*, vol. 16, no. 1, pp. 493–512, 2014.
[16] Open Networking Foundation. Available online: https://www.opennetworking.org, 2013.
[17] P. Manso, J. Moura, and C. Serro, "SDN-based intrusion detection system for early detection and mitigation of DDoS attacks," *Information*, vol. 10, pp. 106, 2019.
[18] N. Gude et al., "NOX: Towards an operating system for networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 3, pp. 105–110, 2008.
[19] POX Network Software Platform. n.d. https://github.com/noxrepo/nox
[20] Ryu SDN Framework. Ryu. https://osrg.github.io/ryu/
[21] Floodlight SDN OpenFlow Controller. n.d. https://github.com/floodlight/floodlight
[22] Trema. n.d. https://github.com/trema/trema
[23] OpenDaylight Foundation. n.d. OpenDayLight. http://www.opendaylight.org/
[24] Open Network Operating System. n.d. https://github.com/opennetworkinglab/onos
[25] G. Wang, Y. Zhao, J. Huang, and Y. Wu, "An Effective Approach to Controller Placement in Software Defined Wide Area Networks," *IEEE Transactions on Network and Service Management*, vol. 15, no. 1, 2018.
[26] P. Bispo, D. Corujo, and R. L. Aguiar, "A qualitative and quantitative assessment of SDN controllers," in *Young Engineers Forum (YEF-ECE) 2017 International*, pp. 6–11, 2017.
[27] R. Izard, and H. Akcay, "Floodligh web GUI," https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/40403023/Web+GUI, 2017.

[28] R. Khondoker, A. Zaalouk, R. Marx, and K. Bayarou, "Feature-based comparison and selection of software defined networking (SDN) controllers," in *2014 World Congress on Computer Applications and Information Systems (WCCAIS)*, pp. 1–7, 2014.
[29] O. Belkadi, and Y. Laaziz, "A systematic and generic method for choosing A SDN controller," *International Journal of Computer Networks and Communications Security*, vol. 5, no. 11, pp. 239–247, 2017.
[30] A. Abdelaziz, A. T. Fong, A. Gani, U. Garba, S. Khan, A. Akhunzada, H. Talebian, and K. W. R. Choo, "Distributed controller clustering in software defined networks," *PLoS One*, 2017.
[31] J. Huang, J. Zou, and C. C. Xing, "Competitions among service providers in cloud computing: A new economic model," *IEEE Transactions on Network and Service Management*, vol. 15, no. 2, pp. 866–877, 2018.
[32] J. Huang, Q. Duan, S. Guo, Y. Yan, and S. Yu, "Converged network-cloud service composition with end-to-end performance guarantee," *IEEE Transactions on Cloud Computing*, vol. 6, no. 2, pp. 545–557, 2018.
[33] D. Anderson, "An investigation into the use of software defined networking controllers in aerial networks," in *IEEE Military Communications Conference (MILCOM)*, 2017.
[34] O. Salman, I. H. Elhajj, A. Kayssi, and A. Chehab, "SDN controllers: A comparative study," in *2016 18th Mediterranean Electrotechnical Conference (MELECON)*, pp. 1–6, 2016.
[35] Y. Zhao, L. Iannone, and M. Riguidel, "On the performance of SDN controllers: A reality check," in *IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*, 2015.
[36] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov, and R. Smeliansky, "Advanced study of SDN/OpenFlow controllers," in *Proceeding 9th Central Eastern European Software Engineering Conference Russia (CEE-SECR)*, 2013.
[37] A. L. Stancu, S. Halunga, A. Vulpe, G. Suciu, O. Fratu, and E. C. Popovici, "A comparison between several Software Defined Networking controllers," in *12th International Conference on Telecommunication in Modern Satellite, Cable and Broadcasting Services (SIKS)*, pp. 223–226, 2015.
[38] K. Kaur, S. Kaur, and V. Gupta, "Performance analysis of python based OpenFlow controllers," in *EEECOS*, 2016.
[39] I. Z. Bholebawa, and U. D. Dalal, "Performance analysis of SDN/OpenFlow controllers: POX versus floodlight," *Wireless Personal Communications*, vol. 28, no. 2, pp. 1679–1699, 2018.
[40] H. Shiva, and C. G. Philip, "A comparative study on software defined networking controller features," *International Journal of Innovative Research in Computer and Communication Engineering*, vol. 4, no. 4, 2016.
[41] V. R. S. Raju, "SDN controllers comparison," in *Proceedings of Science Globe International Conference*, 2018.
[42] D. Sakellaropoulou, "A qualitative study of SDN Controllers," https://mm.aueb.gr/master_theses/xylomenos/Sakellaropoulou_2017.pdf, 2017.
[43] A. A. Semenovykh, and O. R. Laponina, "Comparative analysis of SDN controllers," *International Journal of Open Information Technologies*, vol. 6, no. 7, 2018.
[44] G. A. Miller, "The magical number seven, plus or minus two some limits on our capacity for processing information," *Psychological Review*, vol. 63, no. 2, pp. 81–97, 1956.
[45] D. Erickson, "The beacon OpenFlow controller," in *Proceedings of ACM SIGCOMM Workshop on Hot Topics in Software-defined Networking (HotSDN)*, 2013.
[46] Z. Cai, A. L. Cox, and T. S. Ng, "Maestro: A system for scalable OpenFlow control," Tech. Rep. TR10-08, Rice University, 2010.
[47] http://opennetworkingfoundation.github.io/libfluid/
[48] L. Mamushiane, A. Lysko, and S. Dlamini, "A comparative evaluation of the performance of popular SDN controllers," in *10th Wireless Days Conference (WD)*, 2018.
[49] A. Ishizaka, and P. Nemery, *Multi-criteria decision analysis: methods and software*, John Wiley & Sons, 2013.

[50] G. Buyukozkan, C. Kabraman, and D. Ruan, "A fuzzy multi-criteria decision approach for software development strategy selection," *International Journal of General Systems*, vol. 33, pp. 259–280, 2004.

[51] G. A. Mendoza, and H. Martins, "Multi-criteria decision analysis in natural resource management: A critical review of methods and new modelling paradigms," *Forest Ecology and Management*, pp. 1–22, 2006.

[52] X. Yan, P. Dong, T. Zheng, and H. Zhang, "Fuzzy and Utility Based Network Selection for Heterogeneous Networks in High-Speed Railway," in *Wireless Communications and Mobile Computing*, 2017.

[53] P. Boateng, Z. Chen, and S. O. Ogunlana, "An analytical network process model for risks prioritisation in megaprojects," *International Journal of Project Management*, vol. 33, no. 8, pp. 795–811, 2015.

[54] S. Nazir, S. Anwar, S. A. Khan, S. Shahzad, M. Ali, R. Amin et al., "Software component selection based on quality criteria using the analytic network process," *Abstract and Applied Analysis*, 2014.

[55] H. Farman, H. Javed, B. Jan, J. Ahmad, S. Ali, F. N. Khalil, and M. Khan, "Analytical network process based optimum cluster head selection in wireless sensor network," *PLoS One*, vol. 12, no. 7, 2017.

[56] S. Sun, Y. Wu, Y. Ma, L. Wang, Z. Gao, and C. Xia, "Impact of degree heterogeneity on attack vulnerability of interdependent networks," *Scientific Reports*, 2016.

[57] T. L. Saaty, *Decision making with dependence and feedback: The analytic network process*, RWS Publications, 2001.