Journal of Computer Science and Engineering Research (JCSER)
Volume 2, Issue 2, Date: 31 - December – 2025
ISSN: 2978-4328, DOI: 10.64820/AEPJCSER.22.1.7.122025

Graph Detection Time of Software Defined Network

Salem Omar Sati
Faculty of Information Technology, University of Misurata, Libya
Email: salem.sati@it.misuratau.edu.ly

***Abstract*—Software Defined Networks (SDNs) propose the decoupling of the control plane from the forwarding plane. To implement this new architecture, the OpenFlow SDN control protocol can be employed for communication between virtual switches. This approach enables centralized management and monitoring. However, the development of SDN technology necessitates appropriate testing and measurement tools to assess performance. Currently available open-source performance tools for SDN architecture primarily focus on basic attributes, overlooking the evaluation of numerous SDN controllers in terms of Graph Detection time. This study aims to address this gap by introducing a performance monitoring tool for SDN controllers, which evaluates two Javabased SDN controllers. The tool measures the Graph Detection speed across various network sizes using a Python-based northbound interface. The evaluation includes popular controllers like Open Network Operating System and OpenDaylight, assessing their performance with dedicated control channels. Simulation results indicate that OpenDaylight outperforms Open Network Operating System in terms of Graph Detection time. Index Terms—Graph Detection Time, ODL Controller, ONOS Controller, Rest API, Dedicated Control Channel, Control Northbound Interface.**

Keywords—Graph; Detection; Time; Software; Defined Network.

I. INTRODUCTION

SDN Software Defined Networking (SDN) [1] is an innovative technology that brings forth new network requirements. In SDN networks, control is logically centralized in a key component called the controller. Among the various protocols used in SDN networks, the OpenFlow [2] protocol stands out as the most commonly employed. The SDN controller provides different signaling protocols and standards for establishing connections with virtual switches. The OpenFlow protocol, known for its openness, is often used by controllers to connect with virtual switches. Numerous SDN controllers have been proposed by different vendors, utilizing various programming languages and exhibiting distinct characteristics. For instance, there is the NOX [3] open-source controller, implemented in C++ by Nicira, which serves as a basis for the POX [4] controller a Python implementation. Another Python-based controller is RYU [5]. On the Java side, there are three SDN controllers: Open Network Operating System (ONOS) [6], OpenDayLight (ODL) [7], and Floodlight [8]. With so many controller options available, selecting the most suitable one can be challenging for SDN designers. Moreover, they may question *which controller ONOS or ODL offers faster Graph Detection time*. To aid designers in making informed decisions, a comprehensive and programmable evaluation is necessary. This evaluation should measure SDN performance based on Graph Detection time, providing valuable insights. In this research, we propose and implement a Python performance monitoring software for SDN controllers, utilizing the OpenFlow protocol for communication and control. Irrespective of network size, this software enables managers to measure the Graph Detection time of SDN controllers. Specifically, we assess the Graph Detection time of ONOS and ODL controllers, and the results indicate that ODL outperforms ONOS in terms of Graph Detection time. The structure of this paper is as follows: Section II discusses related topics and previous research. Section III introduces SDN concepts, while Section IV presents various types of SDN control channels. In Section V, we delve into the features of the OpenFlow protocol. Section VI highlights popular controllers and their deployment to validate the implemented tool. Section VII explores different network sizes used to compare the SDN controllers. The architecture and functionalities of the proposed software are outlined in Section VIII. The numerical results collected from the test scenario are discussed in Section IX. Finally, Section X outlines future directions for this research.

II. RELATED WORKS

Numerous research papers have compared SDN controllers in terms of architecture and efficiency, highlighting their respective strengths and weaknesses. For instance, in one paper [9], the authors evaluated two distributed OpenFlow controllers, ONOS and OpenDaylight, using the Cbench tool. The study measured throughput, latency, and software scalability of these Java-based controllers in both real-world and virtualized environments. The results indicated that ONOS outperformed OpenDaylight in terms of higher throughput and reduced latency. Another paper [10] focused on comparing two Javabased controllers, OpenDaylight (ODL) and Open Network Operating System (ONOS). The simulation results showed that the performance of ODL and ONOS in SDN was affected by the increasing number of OpenFlow switches. However, the study found that ONOS exhibited better performance metrics than ODL as the number of switches increased. In another study [11], the performance of POX and Floodlight controllers was analyzed, considering throughput and latency. The evaluation utilized the Mininet

Received: 27-12-2024
Revised: 21-9-2025
Published: 31-12-2025
This article is freely accessible under the Creative Commons Attribution License, allowing for its use, distribution, and reproduction in any format, as long as the original work is correctly cited. © 2025 The Authors.

emulator and examined four topologies for controller-to-controller comparison. The findings demonstrated that the Floodlight controller outperformed the POX controller in terms of latency and throughput. A study [12] compared the POX and RYU controllers, both open-source SDN controllers.

The results showed that the RYU controller outperformed the POX controller in terms of throughput, delay, and overhead. In a different work [13], the OpenDaylight and Floodlight controllers were evaluated in terms of packet loss and latency. The study found that the OpenDaylight controller performed better in terms of latency, while the Floodlight controller exhibited better performance under heavy traffic loads. Another study [14] examined the RYU SDN, Floodlight, and ONOS controllers, providing unique perspectives on performance and scalability when measuring throughput and topology discovery time. In a paper [15] focusing on cloud computing, a containerized architecture capable of scaling up and down based on traffic load was proposed for the SDN control plane. The results showed that the proposed system effectively handled high incoming new-flow requests and optimized resource efficiency through a traffic-adaptive algorithm. A study [16] suggested a method for monitoring software-defined data planes in massively distributed SDN networks, demonstrating improved performance with the suggested strategy. In a paper [17], the authors proposed a Packet-In filtering mechanism that categorized Packet-In messages and forwarded only the necessary ones to other core controller modules. Lastly, a study [18] investigated the performance of SDN controllers in data center networks. The findings indicated that the POX controller performed better than RYU in data center environments. This paper considers both ONOS and ODL controllers which are offer capabilities for efficient Graph Detection and support for out-of-band (dedicated) control plane deployments. However, the actual Graph Detection time and effectiveness of the out-of-band control plane can be influenced by various factors, including network size, topology, and configuration. It is recommended to evaluate and test the controllers in the specific network environment to determine their performance in terms of Graph Detection time and out-of-band control plane functionality.

III. SOFTWARE DEFINED NETWORKING (SDN)

Data communication networks require management and maintenance to ensure the delivery of critical services. To enhance network management and serviceability, SDN was introduced as a technology that separates the control plane from the data plane. The data plane is responsible for data forwarding, while the control plane, residing in a centralized component called the SDN controller, makes decisions for SDN switches. The SDN controller responds to requests from connected SDN switches and is programmed via API interfaces. The control plane operations, such as routing and security, are performed by the SDN controller, which communicates with the switches through established protocols using the southbound API interface. The OpenFlow protocol is widely used for SDN control. The controller manages the network's flow tables, while the northbound API is used for managing the SDN controller itself. This programmable interface simplifies network communication and allows developers to create applications without needing to understand the underlying functionality of SDN devices. The availability of API interfaces in SDN abstracts the complexities of the underlying network infrastructure, enabling designers to focus on developing and deploying new SDN applications. These applications can utilize the OpenFlow protocol and configure it to meet specific performance requirements in the SDN application layer. This flexibility allows for easier development and deployment of SDN applications tailored to specific needs. SDN controllers often provide REST APIs (Representational State Transfer Application Programming Interfaces) as a means of communication and interaction with the controller. REST APIs follow the principles of the REST architectural style, allowing clients to access and manipulate resources using standard HTTP methods. There are some common features and functionalities provided by the REST APIs of SDN controllers. It's important to note that the specific features and capabilities of the REST APIs may vary between different SDN controllers. It's recommended to refer to the documentation and resources provided by the particular SDN controller to understand the available REST API endpoints, request formats, and response structures.

IV. SDN CONTROL PLAN

The context of SDN, the control plane can be implemented using either an in-band or out-of-band approach as shown in Figure 1 (a) and (b).. These terms refer to how the control plane traffic is separated from the data plane traffic within the network. *In-Band Control Plane:* In an in-band control plane as shown in Figure 1 (a), the control plane traffic and the data plane traffic share the same network infrastructure. Control messages and data packets travel over the same physical or logical paths. This approach simplifies the network architecture but can introduce potential performance and security challenges, as control and data traffic compete for network resources. *Outof-Band Control Plane:* In an out-of-band control plane as shown in Figure 1 (b), the control plane traffic is separated from the data plane traffic and is carried over a dedicated network infrastructure. This infrastructure can be a separate physical network or logically isolated paths within the network. By keeping control plane traffic separate, it provides better isolation, scalability, and security. However, it may introduce additional complexity in terms of network architecture and management. The selection between in-band and out-of-band control plane depends on various factors, including network requirements, scalability needs, performance considerations, and security concerns. In-band control planes are simpler to deploy and manage, but they may suffer from limitations in terms of scalability and security. Out-of-band control planes offer better isolation and security, but they come with added complexity and potential resource overhead.

V. OPEN FLOW CONTROL PLANE PROTOCOL

OpenFlow protocol is an open standard signaling protocol designed as a communication rule between SDN devices. OpenFlow was developed by Stanford University in the year 2008 [19]. SDN constructed as a centralized form of the control plane and it generates multiple packet flows using programmable properties specified by a high abstraction.

OpenFlow is a protocol widely used in SDN to enable communication between the SDN controller and the network switches.the first version was OpenFlow 1.0. This was the initial version of the OpenFlow protocol, released in December 2009. It introduced the concept of flow tables and flow-based forwarding, allowing for centralized control of network flows. Where OpenFlow 1.7 was the last version.

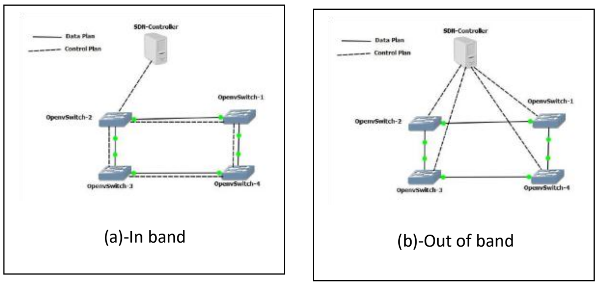

Fig. 1. SDN Control Plan Types

This version, released in October 2017, OpenFlow 1.7 introduced support for large-scale deployments with features like flow object tables, packet replication and cloning, and advanced metering functionalities. There are several OpenFlow control message types used for communication between the SDN controller and the network switches. These message types facilitate the control and management of the network flows. They are considered as a common OpenFlow message types used in the SDN control plane:

1) Hello: The Hello message is used for establishing a connection between the controller and the switch. It is the first message exchanged during the handshake process.
2) Features Request and Reply: The Features Request and Features Reply messages are used to exchange information about the capabilities and features supported by the switch. The controller requests the switch's features, and the switch responds with its capabilities.
3) Set Config: The Set Config message is used to configure the switch's behavior, such as setting the flow table miss behavior, enabling or disabling packet flooding, or configuring other switch-specific parameters.
4) Flow Mod: The Flow Mod message is used to add, modify, or delete flow entries in the switch's flow table. It allows the controller to specify the matching criteria, actions, and duration for the flow entries.
5) Packet-In: The Packet-In message is triggered when a packet arrives at the switch but doesn't match any existing flow entry. It is sent to the controller, allowing it to decide how to handle the packet.
6) Packet-Out: The Packet-Out message is used by the controller to instruct the switch to send a packet out of a specific port. It allows the controller to take control of packet forwarding decisions.
7) Barrier Request and Reply: The Barrier Request and Barrier Reply messages are used to ensure message synchronization between the controller and the switch. The Barrier Request is sent by the controller, and the switch responds with a Barrier Reply once it has processed all previous messages.
8) Statistics Request and Reply: The Statistics Request and Statistics Reply messages are used to retrieve various statistics from the switch, such as flow table statistics, port statistics, or queue statistics.

VI. ODL AND ONOS CONTROLLERS

In this study, a software was developed to evaluate the network Graph Detection time of SDN controllers. The OpenDaylight (ODL) and ONOS controllers were chosen as the implementation platforms due to their widespread adoption in the industry and extensive feature support. These controllers are commonly utilized by various vendors and companies in different projects and solutions, making them commercially competitive options for large-scale and enterprise environments.

A. OpenDaylight (ODL)

OpenDaylight (ODL) is a widely adopted open-source SDN controller developed in collaboration with renowned commercial companies under the Linux Foundation. Networking industry giants like Cisco Systems, Dell, HP, and others have contributed to its development, leveraging their extensive experience in the field. ODL offers various updated distributions to cater to different requirements. One of ODL's notable features is its ability to facilitate enhanced interaction between applications and controllers. It achieves this by leveraging Karaf, a platform that provides all the necessary features and services to build servers based on the Open Service Gateway Initiative (OSGi) standard. With Karaf, users and developers can selectively choose the desired functionalities from a list, allowing for customization of the ODL controller. The ODL controller does not require specific functionalities upon installation. Instead, users can configure it using Karaf to include the functionalities they need for their specific use cases. The ODL controller boasts a southbound API interface, which establishes a connection between the controller and the underlying virtual switches in the network topology. It supports multiple control protocols, such as OpenFlow versions 1.0 and 1.3, .

B. Open Network Operating System (ONOS)

The ONOS controller is implemented in Java and serves as a platform for multi-domain SDN applications. It offers various interfaces, including a REST API, a Command Line Interface (CLI), and a web-based graphical user interface (GUI) that is extensible and user-friendly. The architecture of the ONOS controller consists of different layers. At the top layer are the business applications that leverage the controller's capabilities. In the middle layer, the distributed software core is situated. This core enables the separation of the data plane and control plane, providing a logically centralized view of SDN control. It serves as the interface for accessing the control plane and is decoupled from the overlay and underlay boundaries using Southbound and Northbound API interfaces. The Southbound API interface, facing the south, allows the ONOS controller to interact with the virtual switch environment. The core of the ONOS controller operates based on control plane protocols like OpenFlow, NETCONF, and OVSDB. Through its provider applications, which are based on OSGi components, the ONOS controller interacts with the underlying virtual switches. These provider applications can be enabled or disabled at runtime, and their primary purpose is to manage, control, and configure specific OpenFlow or NETCONF-compatible devices. Additionally, the ONOS controller communicates with underlay devices using various control and management plane protocols, including OpenFlow, NETCONF, and SNMP.

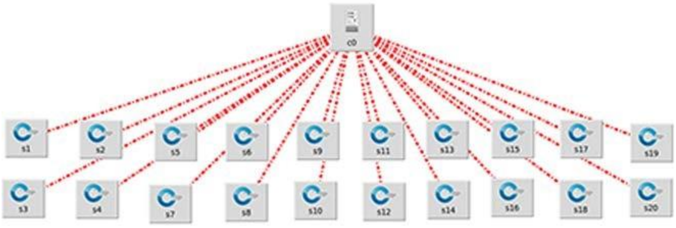

Fig. 2. Scalability Topology of 20 Switches (Small)

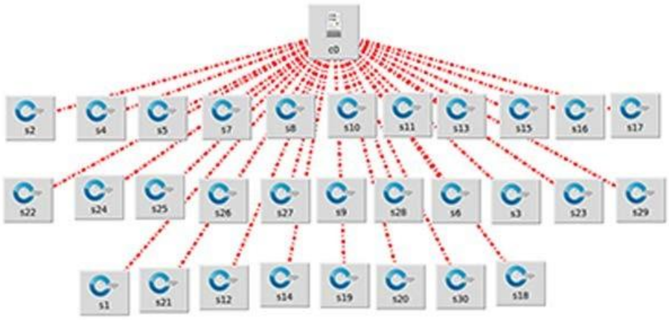

Fig. 3. Scalability Topology of 30 Switches (Medium)

VII. SCALABILITY NETWORKS

This scalability topology comprises multiple virtual switches connected solely to the SDN controller through an out-of-band control channel. The topology, illustrated in Fig 2, Fig 3, and Fig 4, serves as a testbed for measuring control plane scalability. The maximum number of connected virtual switches represents the maximum capacity of the SDN out-of-band control plane's dedicated channel. This measurement is described in the scalability test scenario outlined in RFC 8456 [20], which evaluates the performance of the SDN out-of-band control plane. The topology consists of two dimensions: the network topology based on the number of switches and the performance of the controller's southbound interface and control channel capacity. For this topology, the number of control channels must be equal to the number of switches connected to the SDN controller. Importantly, in this topology, there is no data plane connectivity between the switches. Fig 2 depicts the controller connected to 20 virtual switches, establishing 20 dedicated OpenFlow channels. These channels facilitate the transfer of OpenFlow control messages mentioned in section V. When testing an out-of-band control plane in an SDN environment, connecting virtual switches exclusively to the SDN controller helps isolate and evaluate the effectiveness of the dedicated control channel. This isolation is achieved by connecting the virtual switches solely to the SDN controller, creating a dedicated control channel separate from the data plane. This setup allows for focused testing of the out-of-band control plane's functionality without interference from data plane traffic. Evaluating the control plane is an important aspect of SDN. Testing the outof-band control plane aims to assess its performance, security, and reliability.

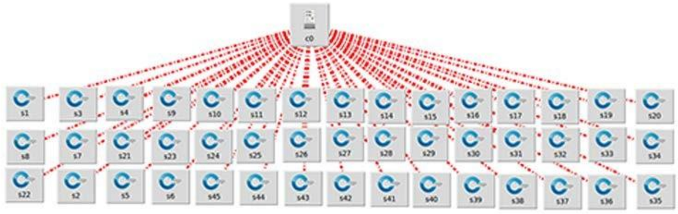

Fig. 4. Scalability Topology of 45 Switches (Large)

By connecting virtual switches solely to the SDN controller, it becomes possible to evaluate how well the control plane operates independently of the data plane. This includes examining controller responsiveness, flow provisioning, and network management capabilities. Furthermore, this setup minimizes interference between the control and data planes. In an out-of-band control plane configuration, the dedicated control channel ensures that control traffic remains unaffected by data plane congestion or disruptions. This isolation helps assess the control plane's resilience and its ability to maintain consistent communication with the virtual switches. Finally, this topology facilitates troubleshooting and debugging. If issues arise during testing, the clear separation between the control plane and data plane connections simplifies the process of identifying and resolving problems. Administrators can focus on the control channel configuration and communication between the SDN controller and virtual switches, streamlining troubleshooting efforts. Based on these considerations, we can modify the scalability topology to include 30 and 45 virtual switches, as illustrated in Fig 3 and Fig 4 respectively.

VIII. SOFTWARE PROPOSED FOR MEASURING GRAPH DETECTION TIME

This research paper introduces a software designed for evaluating the performance of OpenFlow SDN controllers. Within the realm of SDN test tools, an open-source tool called "Cbench" is commonly used to test OpenFlow controllers. Cbench provides test results for measuring the throughput and latency of OpenFlow controllers. However, Cbench has not been updated or supported recently, as it only deals with OpenFlow version 1.0 and does not measure the Graph Detection time taken by the controller to detect the network topology. To address these limitations, the paper proposes a software that collects topology information from the northbound SDN controller. This software aims to benchmark OpenFlow controllers using the REST northbound API. It is compatible with OpenFlow version 1.3 and follows an architecture based on a standalone centralized controller model. The software is implemented using the Python programming language due to its flexibility and the availability of extensive documentation. Figures 5 and 6 in the paper showcase the Python code for measuring Graph

```python
class ODL:

    def __init__(self):
        self.baseUrl = ('http://IP:Port/restconf')
        self.h = httplib2.Http(".cache")
        self.h.add_credentials('admin', 'admin')

    def get_switches_number(self):
     try:
      url = self.baseUrl + '/operational/opendaylight-inventory:nodes'
      logging.debug('url %s', url)
      _,content = self.h.request(url, "GET")
      allContent = json.loads(content)
      node_list=allContent['nodes']
      if node_list == {}:    ### if topology is empty return 0
       return 0
      return (len(node_list['node'])) ### return number of items in nodes (switches) list
     except:
      return 0


    def getConvergenceTime(self):
     try :
      nodes = self.get_switches_number()
      max_nodes = 20

      if(nodes != 0):  ### first request must return 0 switches to store start time
        return 0

      while(nodes < max_nodes):

         if(nodes == 0): ### storing start time
             self.start_time = int(time.time() * 1000)

         if(nodes >= max_nodes): ### storing endtime
           self.end_time = int(time.time() * 1000)

         nodes = self.get_switches_number()

      return int(self.end_time - self.start_time)
     except Exception as e:
      return 0
```

```python
class ONOS:

    def __init__(self):
        self.baseUrl = ('http://IP:Port/onos/v1')
        self.h = httplib2.Http(".cache")
        self.h.add_credentials('onos', 'rocks')

    def get_switches_number(self):
      url = self.baseUrl + '/devices'
      logging.debug('url %s', url)
      _,content = self.h.request(url, "GET")
      allContent = json.loads(content)
      node_list=allContent['devices']

      counter = 0
      for switch in node_list:
        if(switch['available'] == True):
          counter = counter+1
      return (counter)


    def getConvergenceTime(self):
     try :
      nodes = self.get_switches_number()
      max_nodes = 20

      if(nodes != 0):  ### first request must return 0 switches to store start time
        return 0

      while(nodes < max_nodes):

         if(nodes == 0): ### storing start time
             self.start_time = int(time.time() * 1000)

         if(nodes >= max_nodes): ### storing endtime
           self.end_time = int(time.time() * 1000)

         nodes = self.get_switches_number()

      return int(self.end_time - self.start_time)
     except Exception as e:
      return 0
```

Fig. 5. Graph Detection Time Calculation For ODL Controller.

Detection time for ODL and ONOS controllers, respectively. The proposed software communicates with the specified SDN controller that is connected to determine the current number of virtual switches. These virtual switches are connected to the OpenFlow controller through an out-of-band control plane channel. If the software detects that the current number of switches is zero, it indicates that the Mininet emulator and its topology are not connected to the SDN controller's virtual machine. This implies that the SDN controller is not yet connected to the Mininet emulator. In such cases, the software initiates the Graph Detection time measurement by storing the current response time in milliseconds format. Next, the software determines the maximum number of switches and connects the topology with the Mininet emulator, which is already connected to the controller. The software monitors the virtual switches detected by the controller through the northbound API interface. It waits until the current number of detected switches reaches the total number of switches specified by the maximum number of switches variable set at the beginning. At this point, the software records the last response time, which corresponds to the detection of the last switch in the topology, also in milliseconds format. Finally, the software calculates the precise Graph Detection time of the tested topology by subtracting the first switch detection time from the last switch detection time in milliseconds. The Python code for the ODL API is illustrated in Figure 5.

The ODL controller offers a wide range of API endpoints that provide flexibility to controller application programmers. In the proposed software, the topology inventory endpoint of the ODL controller is utilized. This endpoint provides the software with brief information about the currently connected switches in the topology. Additionally, the ODL controller has default login credentials (admin for both the username and password), which are included in the API request header, as shown in Figure 5. The software receives the response in JSON format and counts the number of nodes (i.e., detected switches) in the response to determine the Graph Detection time. The computed result of the Graph Detection time is then displayed by the software.

Figure 5 presents the Python code for the northbound API of the ONOS controller. The ONOS Controller API is simpler compared to the ODL controller API. The login credentials for the ONOS controller are as follows: ”onos” for the username and ”rocks” for the password. These credentials are included in Figure 5: Graph Detection Time Calculation For ONOS Controller. The request header of the ONOS controller API, as depicted in Figure 5. The response from the software is received in JSON format. When interacting with the ONOS controller, the software utilizes an endpoint that provides the response containing the state of connected switches. This response either includes a count or a list of switches that are currently connected or have been connected to the ONOS controller at any given time. To count the detected switches in the topology, the software uses the ”available” attribute, as illustrated in Figure 5. This attribute indicates whether the ONOS controller is currently connected (true) or not (false). The final section of the code in Figure 5 shows the counter that keeps track of the total number of detected switches in the topology. These switches are connected to the ONOS controller through an out-of-band control plane using an OpenFlow version 1.3 control channel.

IX. GRAPH DETECTION TIME SOFTWARE RESULTS

This research paper focuses on measuring the Graph Detection time of SDN controllers, specifically comparing the ONOS and ODL controllers. To perform the measurements, a software tool is developed using Python, which interacts with the controllers' APIs to collect the necessary data. The comparison between the controllers is conducted in terms of unconnected switches and the scalability of the topology. Three different sizes of topologies are considered: 20 switches (Small), 30 switches (Medium), and 45 switches (Large), as depicted in Figure 6. According to the results presented in Figure 6, the ODL controller exhibits the lowest Graph Detection time. For the Small topology with 20 switches, the ODL controller achieves a Graph Detection time of 0.5 seconds, whereas the ONOS controller takes 0.7 seconds. As the topology size increases to the Large configuration with 45 switches, the ODL controller's Graph Detection time reaches one second, while the ONOS controller's Graph Detection time is 1.7 seconds. Furthermore, Figure 6 also reveals that the ODL controller outperforms the ONOS controller in terms of discovery time across all scalability scenarios and different topology sizes.

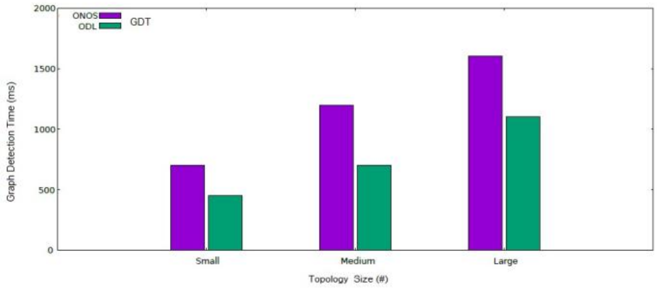

Fig. 6. Software Graph Detection Time Results.

X. CONCLUSION AND FUTURE WORK

This research paper introduces a software tool that evaluates the Graph Detection time of ONOS and ODL controllers. The software is developed using the Python programming language and is designed to assess the controllers in standalone mode. The testing involves emulating a topology using Mininet with virtual OpenFlow switches. To validate the software, the researchers conducted topology testing using well-known Java SDN controllers, including ONOS and ODL. With the proposed software, administrators and designers can measure the Graph Detection time of the out-of-band or dedicated SDN controller plane. The software offers simplicity in usage and provides Graph Detection time based on the responses received from the northbound API of the controllers. The results obtained from the software's measurements demonstrate that ODL performs better than ONOS in terms of Graph Detection time across different scalability scenarios with topologies consisting of 20, 30, and 45 virtual switches (Vswitches). As the topology size increases from the Small configuration of 20 switches to the Medium configuration of 30 switches, the Graph Detection time of the ODL controller increases by 20%, whereas for the ONOS controller, it increases by 80%. However, when further increasing the topology size from the Medium configuration of 30 switches to the Large configuration of 45 switches, the ODL controller's Graph Detection time increases by 30%, while the ONOS controller's Graph Detection time increases by 60%. Consequently, it can be concluded that the ODL controller outperforms the ONOS controller in terms of Graph Detection time.As future work, the authors plan to extend their testing to include other SDN controllers.

REFERENCES

[1] F. Xu, J. He, and X. Wu, "First study on supply and demand network with multi-functional and opening characteristics for enterprise (SDN)," in *Proceedings of the IEEE International Conference on Systems, Man & Cybernetics: The Hague, Netherlands, 10-13 October 2004*. IEEE, 2004, pp. 2108–2113.

[2] R. Narisetty, L. Dane, A. Malishevskiy, D. Gurkan, S. Bailey, S. Narayan, and S. Mysore, "Openflow configuration protocol: Implementation for the of management plane," in *2013 Second GENI Research and Educational Experiment Workshop, Salt Lake City, UT, USA, March 20-22, 2013*. IEEE Computer Society, 2013, pp. 66–67.

[3] G. Yao, J. Bi, and P. Xiao, "Source address validation solution with openflow/nox architecture," in *Proceedings of the 19th annual IEEE International Conference on Network Protocols, ICNP 2011, Vancouver, BC, Canada, October 17-20, 2011*. IEEE Computer Society, 2011, pp. 7–12.

[4] R. Jmal and L. C. Fourati, "Implementing shortest path routing mechanism using openflow POX controller," in *The International Symposium on Networks, Computers and Communications, ISNCC 2014, Hammamet, Tunisia, June 17-19, 2014*. IEEE, 2014, pp. 1–6.

[5] S. Bhardwaj and S. N. Panda, "Performance evaluation using RYU SDN controller in software-defined networking environment," *Wirel. Pers. Commun.*, vol. 122, no. 1, pp. 701–723, 2022.

[6] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, and G. M. Parulkar, "ONOS: towards an open, distributed SDN OS," in *Proceedings of the third workshop on Hot topics in software defined networking, HotSDN '14, Chicago, Illinois, USA, August 22, 2014*, A. Akella and A. G. Greenberg, Eds. ACM, 2014, pp. 1–6.

[7] J. Medved, R. Varga, A. Tkacik, and K. Gray, "Opendaylight: Towards a model-driven SDN controller architecture," in *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks, WoWMoM 2014, Sydney, Australia, June 19, 2014*. IEEE Computer Society, 2014, pp. 1–6.

[8] X. Chen, D. Guo, W. Ma, and L. He, "Floodsight: A visual-aided floodlight controller extension for SDN networks," in *Smart Graphics 13th International Symposium, SG 2015, Chengdu, China, August 26-28, 2015, Revised Selected Papers*, ser. Lecture Notes in Computer Science, Y. Chen, M. Christie, and W. Tan, Eds., vol. 9317, 2015, pp. 75–86.

[9] M. Darianian, C. Williamson, and I. Haque, "Experimental evaluation of two openflow controllers," in *25th IEEE International Conference on Network Protocols, ICNP 2017, Toronto, ON, Canada, October 10-13, 2017*. IEEE Computer Society, 2017, pp. 1–6.

[10] M. Sati, M. Emshiheet, A. Majouk, and S. O. Sati, "Control plane comparison ofA opendaylight andˆ A open network operating systemˆ

controllers," in *Asia Pacific Advanced Network*, D. Herath, S. Date, U. Jayasinghe, V. Narayanan, R. Ragel, and J. Wang, Eds. Cham: Springer Nature Switzerland, 2024, pp. 100–111.

[11] I. Z. Bholebawa and U. D. Dalal, "Performance analysis of sdn/openflow controllers: POX versus floodlight," *Wirel. Pers. Commun.*, vol. 98, no. 2, pp. 1679–1699, 2018.

[12] J. Alzarog, A. Almhishi, A. Alsunousi, A. Elasaifer, W. Eltarjaman, and S. O. Sati, "Sdn controllers comparison based on network topology," in *2022 Workshop on Microwave Theory and Techniques in Wireless Communications (MTTW)*, 2022, pp. 204–209.

[13] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM Workshop on Hot Topics in Networks. HotNets 2010, Monterey, CA, USA - October 20 - 21, 2010*, G. G. Xie, R. Beverly, R. T. Morris, and B. Davie, Eds. ACM, 2010, p. 19.

[14] J. B. da Silva, F. S. D. da Silva, E. P. Neto, M. O. O. Lemos, and A. Neto, "Benchmarking of mainstream SDN controllers over open off-the-shelf software-switches," *Internet Technol. Lett.*, vol. 3, no. 3, 2020.

[15] D. A. Khoa, N. T. Kiem, N. T. Kien, N. N. Tuan, and N. H. Thanh, "Traffic-adaptive scheme for SDN control plane with containerized architecture," in *17th International Conference on Ubiquitous Information Management and Communication, IMCOM 2023, Seoul, Korea, Republic of, January 3-5, 2023*, S. Lee, H. Choo, and R. Ismail, Eds. IEEE, 2023, pp. 1–6.

[16] Y. Wang, J. Bi, and K. Zhang, "A tool for tracing network data plane via sdn/openflow," *Sci. China Inf. Sci.*, vol. 60, no. 2, p. 22304, 2017.

[17] S. Ahmad and A. H. Mir, "Protection of centralized SDN control plane from high-rate packet-in messages," *Int. J. Inf. Sec.*, vol. 22, no. 5, pp. 1197–1206, 2023.

[18] J. Alzarog, A. Almhishi, A. Alsunousi, T. A. Abulifa, W. Eltarjaman, and S. O. Sati, "Pox controller evaluation based on tree topology for data centers," in *2022 International Conference on Data Analytics for Business and Industry (ICDABI)*, 2022, pp. 67–71.

[19] M. Boucadair and C. Jacquenet, "Software-defined networking: A perspective from within a service provider environment," *RFC*, vol. 7149, pp. 1–20, 2014.

[20] B. Vengainathan, A. Basil, M. Tassinari, V. Manral, and S. Banks, "Benchmarking methodology for software-defined networking (SDN) controller performance," *RFC*, vol. 8456, pp. 1–64, 2018.