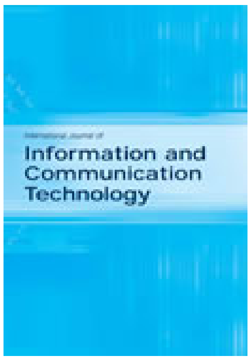


**International Journal of Information and Communication Technology**

ISSN online: 1741-8070 - ISSN print: 1466-6642
https://www.inderscience.com/ijict

# Performance evaluation of SDWNs in cloud systems

Muhammad Salah ud din, Byung-Seo Kim

**DOI:** 10.1504/IJICT.2022.10051129

**Article History:**

| | |
|---|---|
| Received: | 29 January 2021 |
| Last revised: | 03 October 2022 |
| Accepted: | 25 July 2022 |
| Published online: | 26 October 2022 |

# Performance evaluation of SDWNs in cloud systems

Muhammad Salah ud din

Department of Electronics and Computer Engineering,
Hongik University,
Sejong, South Korea
Email: salah_udin@outlook.com

Byung-Seo Kim*

Department of Software and Communications,
Hongik University,
Sejong, South Korea
Email: jsnbs@hongik.ac.kr
*Corresponding author

**Abstract:** The bursty traffic conditions and fast mobility of consumers induce significant delays in wireless networks. Taking the aforementioned factors into account, designing cost-effective wireless solutions with efficient resource usage and manageability is significantly challenging. Although cloud providers undeniably offer high-performance scalable wireless networks, delays may occur due to frequent packet losses as a result of weak signal strength, high mobility, and congestion. There is a slight difference between a wireless networking environment and typical wired networks, however, in practice, various variables exist, making it impossible to fully implement the core advantages of SDN such as rapid data processing, among others. To cope with such situations, we propose software-defined wireless networking (SDWN), a new means for wireless network virtualisation and programming capabilities. Moreover, to evaluate the proposed system, Mininet-WiFi in SDWN is used and conduct an experimental study in terms of packet transmission time and mobility visualisation over distance.

**Keywords:** software defined networking; SDN; software-defined wireless networking; SDWN; Mininet; Mininet-WiFi; controller; OpenFlow; emulator; wireless network; cloud system.

**Reference** to this paper should be made as follows: Salah ud din, M. and Kim, B-S. (2022) ‘Performance evaluation of SDWNs in cloud systems’, *Int. J. Information and Communication Technology*, Vol. 21, No. 4, pp.445–456.

**Biographical notes:** Muhammad Salah ud din received his MS degree in Computer Science (Major in Wireless Communication) from COMSATS University, Islamabad, Pakistan in 2016. He is currently pursuing his PhD in Computer Engineering at the Department of Electronics and Computer Engineering in Graduate School, Hongik University, South Korea. His major interests are in the field of information-centric networking, intelligent ICN for metaverse, vehicular edge/fog computing, wireless sensor networks, internet of things, and underwater wireless sensor networks.

Byung-Seo Kim is working as a Professor at the Department of Software and Communications, Hongik University, South Korea. His research field includes future wireless network and machine learning, IoT, and edge computing.

## 1 Introduction

The core features of software-defined wireless networking (SDWN) are implemented based on software defined networking (SDN), a next-generation network paradigm that manages the network in a centralised manner so that network paths and traffic can be operated and handled freely without human intervention. SDWN separates the hardware from the software in a centralised environment, and the entire network is controlled by the centralised controller. This results in efficient communication, rapid and secure network operations through batch flow rule application, and technology to escape vendor dependencies using Open Interface. To evaluate the performance of the aforementioned networking paradigm, virtual network emulators are actively being used in the research community. These network emulators eliminate the need for the development of real testbed-based implementation for the evaluation. Network emulators support both performance evaluation and debugging of the proposed protocols, and also support various other network-related research problems. Moreover, a network emulator reduces the cost associated with the development of a real testbed. A plethora of networking emulators such as Mininet (Persia et al., 2021), ComNetsEmu (Xiang et al., 2021), etc., are available and among them, the Mininet emulator is widely being used for the evaluation of SDN scenarios. The main advantage of the Mininet emulator is that the developed code can directly be applied to real-world network situations without any modifications and the deployment can be achieved through image files. Mininet-WiFi – which is built on the top of Mininet emulator, is used for the evaluation of real-world wireless network environments. For a cloud infrastructure (Jamil and Kim, 2021) to be built, it is based on the virtualisation of servers or storage. SDWN technology is required for these resources to be effectively connected within wireless networks.

The wireless channel may confront severe fluctuations and never remain stable all the time due to frequent path losses, multipath fading and capture effect, etc., therefore, the user's request may fail to reach the potential destination. Due to a lack of requested reception of requested content, the user may re-initiate the communication to fetch the content. The frequent request transmissions may induce congestion, ignites delays, and intensify the packet losses in the network.

To overcome the abovementioned issues designing cost-effective wireless solutions with efficient resource usage and manageability is the need of time. Taking the intrinsic Mininet feature and with the precise aim to increase the network performance within the limited resource budget and satisfy the consumer requirements within the quality constraints, we propose a state-of-the-art SDWN solution. The main contributions of the proposed work can be summarised as follows:

1 We propose SDWN a new means for wireless network virtualisation and programming capabilities to minimise the delays and network resource usage during bursty traffic conditions.

2 We evaluate the effectiveness of SDWN through extensive simulations in Mininet emulator. Results revealed that SDWN outperformed in terms of packet transmission time and resource utilisation.

The rest of this paper is organised as follows: Section 2 provides background knowledge of SDNs, OpenFlow, emulators in general, Mininet emulator, and Mininet-WiFi emulator. Section 3 conducts an implementation and performance evaluation of packet transmission time and mobility visualisations over distance and finally, Section 4 concludes the paper with future work.

## 2 Background knowledge of SDNs and subtopics

### *2.1 Background knowledge of SDN*

Software-defined networking (Ghaffar et al., 2021) is a network virtualisation approach that is used to optimise the network resources and quickly enable the network to meet business requirements, applications, and traffic, Younus et al. (2019) explained. SDN separates the control plane from the data plane within the network and builds infrastructure by employing software programs. In SDN, the centralised controller is responsible for network orchestration, management, analysis, and automation. As these controllers do not consider networking equipment, they can benefit from scalability, performance, modern cloud computing, and the availability of storage resources. A plethora of SDN controllers is being built by using open platforms and open APIs, enabling orchestration, management, and control of network equipment purchased from different vendors. The separation of control and transport layers increases flexibility and speeds up the launch of new applications. Moreover, the ability of SDN to respond more quickly to problems and outages increases network availability. For this feature implementation, the SDN network structure is divided into three layers. Each layer then communicates with the other through the Open Interface. The Network Control layer to Infrastructure layer interface is called the Southbound API and includes OpenFlow. The interface between the Network Control layer and the Application layer is called the Northbound API. This simplifies the automation of network functions to reduce operational costs, and Table 1 shows the difference between the SDN and the existing network described earlier.

### *2.2 Background knowledge of OpenFlow*

OpenFlow (Yan et al., 2022) corresponds to a communication protocol that resides between the control and the data layer in an SDN environment which defines the content of communication between controllers and hardware equipment and is the basis of SDN technology. It is an open-source project jointly developed by Stanford and UC Berkeley, aiming to separate the forwarding plane and control plane features of a network switch or router and provide protocols for their communication. In addition, OpenFlow's controllers provide technology to determine the path of packets regardless of the vendor of the network. Once the data path is established, OpenFlow provides programmability for network equipment, which defines which network equipment will flow traffic into. One of the reasons for SDN's emergence is the flow of global companies supporting

SDN and OpenFlow. OpenFlow is a component of SDN and is a standard interface for running communication between machines with control and network switches, but there are no restrictions or requirements that must be used within the SDN category. In an SDN environment, it is used to communicate control plane commands to the data plane as a communication protocol between the control plane and the data plane, which can be seen as defining the communication between the controller and the actual or virtual equipment. In addition, policies such as control methods and routing of equipment are defined, and all equipment delivered through this information is stored in the flow table and all of which define the flow of data flow is defined. Flow table consists largely of rule, action, and stats, and in the case of rule, it is an area that defines how and how packets are handled. The flow table's rule consists of 12 tuple, which handles packets with 12 differentiators from layer 1 to layer 4. Action defines how to handle packets defined by the rule. When using the forward command, packets are sent through the specified port, but when using the drop command, packets are discarded. For stats, we show how many packets were matched to the corresponding flow table and how large Byte was transferred. Various transmission information can be found using this information, which is sent to the controller.

**Table 1** Differences between existing networks and SDN

| | *Existing network* | *SDN* |
|---|---|---|
| Network perspective | Hardware centric | Software centric |
| Configuration leadership | Hardware supplied vendor | User |
| Technology openness | Closed structure | Open structure |
| Interlock compatibility | Independent protocol | Standard protocol |
| Management efficiency | Inefficient/high-cost operation | Efficient/reasonable operation |
| The fairness of the market | Monopoly form | Fair competition |
| Embrace new technologies | According to the need for a venter | Acceptance based on user needs |

### *2.3 Background knowledge of controller*

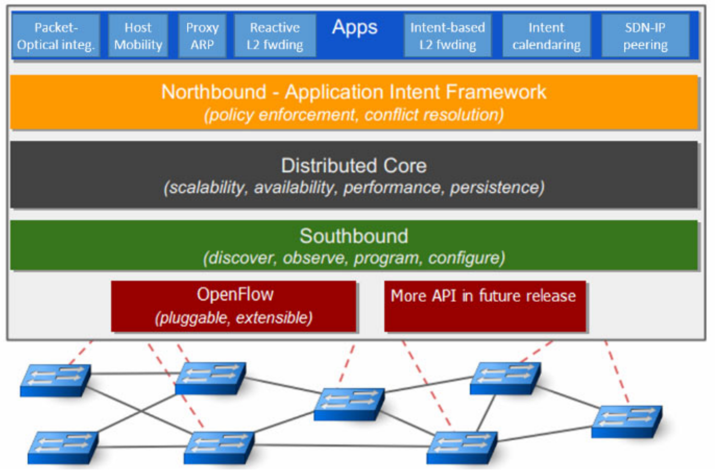
SDN controllers are utilised to control the network. A control point of an SDN network and an application that deploys intelligent networks by managing flow control over applications and business logic. Typically, SDN controller platforms include modules that can perform various network tasks. Some of the basic tasks, including the collection of devices within the SDN network and the collection of each feature, network statistics, etc., can be inserted to support various advanced features, such as improving functionality, performing analysis, and running algorithms that coordinate new rules. Casalicchio and Silvestri (2011) explained the protocols used by SDN controllers to communicate with switches and routers commonly use OpenFlow protocol. Recently, as more SDN networks have been deployed, tasks have been performed between SDN controller domains using common application interfaces such as OpenFlow.

#### *2.3.1 ONOS controller*

ONOS controller was developed by ON. The lab is a non-profit research institute founded in 2012. ONOS Controller was recognised as one of the open-source projects, with

ONOS Project officially participating in the 2015 Linux Foundation. The initial version was developed incorporating two prototypes where the first main target availability and scalability, while the second mainly focused on improving performance. This led to the establishment of the distributed structure of the ONOS controller. Fontes et al. (2015) explained controller requires skills in resource management and delivery, independence of user access, resource virtualisation, abstraction and programming interfaces at the physical and hardware layers, security, and basic mounting of key applications and services. Sameer and Goswami (2018) explained ONOS controller was designed and developed to provide these core functions as a network OS, and as shown below, the basic structure of the ONOS controller was designed for carriers or large network service providers, and the structure of ONOS controller was shown in Figure 1.

**Figure 1** ONOS controller structure (see online version for colours)


Packet-Optical integ.
Host Mobility
Proxy ARP
Reactive L2 fwding
Apps
Intent-based L2 fwding
Intent calendaring
SDN-IP peering
Northbound - Application Intent Framework
(policy enforcement, conflict resolution)
Distributed Core
(scalability, availability, performance, persistence)
Southbound
(discover, observe, program, configure)
OpenFlow
(pluggable, extensible)
More API in future release

*Source:* Lab 08: Multi-Tenant Data Center Network – Part I (n.d.)

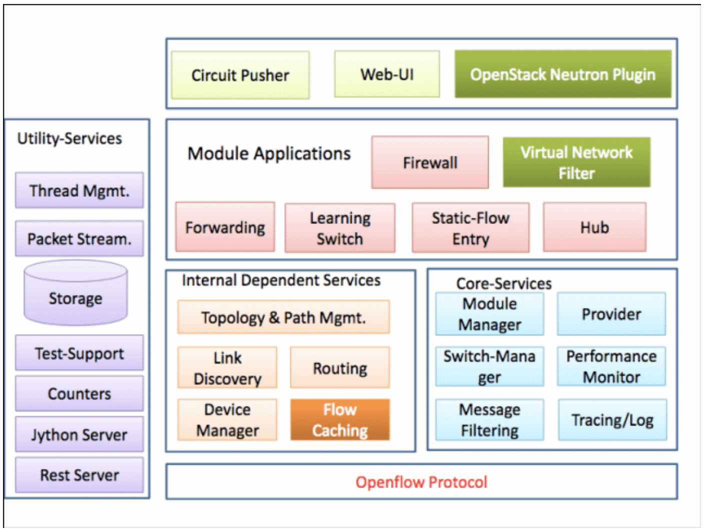
### 2.3.2 *POX controller*

POX controller is a network control platform that provides a high-level programmatic interface for building network operations and control applications. It is composed of Python languages and officially supports OpenFlow 1.0 version and is easy to use. It is a widely used platform for training and research on SDN and network application programming. The branch of the POX controller has two categories: active and release. The active branch is a branch that continues to add new features, and the release branch is a branch that is created when a new version is selected. As simple as configuration, POX controller is a great advantage of being easy to use and has great access to OpenFlow. It can accommodate most of the various applications and can be accessed easily by Python compared to other languages.

### 2.3.3 *Floodlight controller*

Floodlight controller is an SDN controller developed by the community of Big Switch Networks that uses the OpenFlow protocol to coordinate traffic flows in the SDN environment. This controller is a community project originally started on the Big Switch as part of the Opendaylight project and is still sponsored by the big switch network. The Floodlight controller is advantageous for developers because it is written in Java and provides easy software interworking and application development capabilities. The following structure includes REST APIs (Indrawan et al., 2022) that facilitate the programming of products and interfaces and provides coding examples to help developers create products on the Floodlight Web site. As shown in Figure 2 of Saleh Asadollahi and Goswami (2017) the Floodlight controller was tested with physical and virtual OpenFlow compatible switches and supported by network groups of switches that are operational in a variety of environments and that are OpenFlow compatible through non-traditional OpenFlow switches. It is also compatible with OpenStack, which allows the establishment and management of public and private cloud computing platforms and can be run with OpenStack's back end using REST APIs. Furthermore, the controller's performance is excellent because it is highly scalable and provides a simple module loading system. It can be set up with minimal effort and can be processed by mixing OpenFlow network and legacy network.

**Figure 2** Floodlight controller structure (see online version for colours)

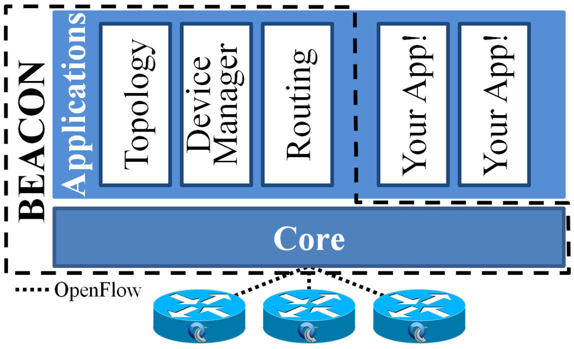


*Source:* Asadollahi and Goswami (2017)

### 2.3.4 *Beacon controller*

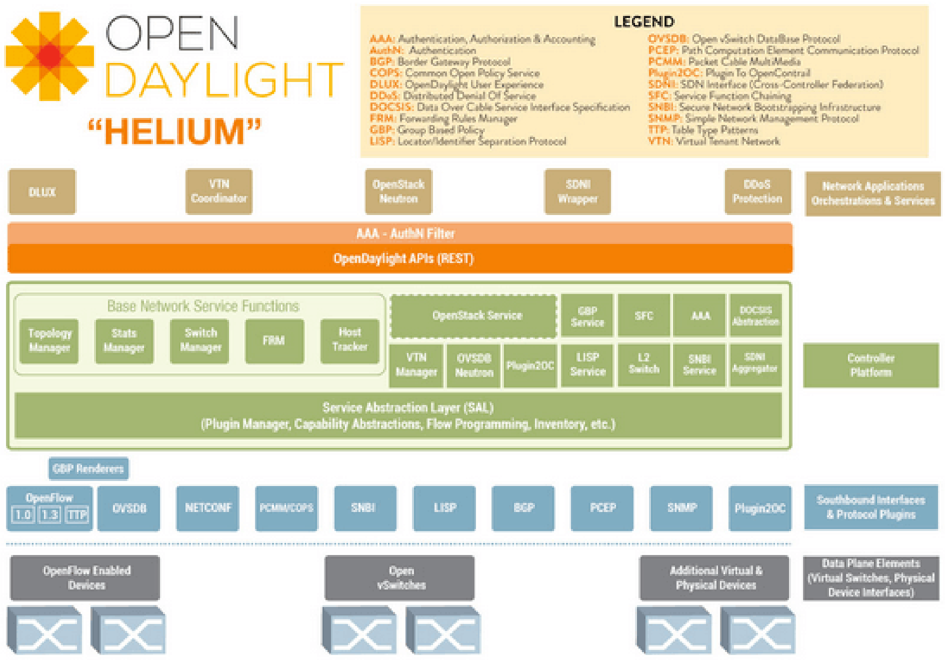
Beacon controller is an open-source project developed by Stanford University in early 2010. According to Erickson (2013), it has been used for several research projects, networking classes, and test deployments, and has been demonstrated by powering more than 100 virtual switches and 20 physical switches, running in experimental data centres. It was developed based on Java and can run on a variety of platforms, from Linux servers to Android phones. According to Siddiqui et al. (2022), Beacon controller's code bundle can start/stop/refresh/install runtime without interrupting other non-dependent bundles (i.e., without disconnecting the switch and changing the running switch or application). It is also easy to start and run, and Java and Eclipse simplify the development and debugging of applications. It is also multi-threaded, capable of identifying performance benchmarks, optionally user-defined scalability with Jetty enterprise web servers, and includes a UI framework. Beacon controller is based on a flexible Java framework, such as Spring and Equinox (OSGi), and is configured as shown in Figure 3 (Erickson, 2013).

**Figure 3** Beacon controller structure (see online version for colours)



*Source:* Erickson (2013)

**Figure 4** Opendaylight controller structure (see online version for colours)


OPEN DAYLIGHT
"HELIUM"
LEGEND
AAA: Authentication, Authorization & Accounting
AuthN: Authentication
BGP: Border Gateway Protocol
COPS: Common Open Policy Service
DLUX: OpenDaylight User Experience
DDoS: Distributed Denial Of Service
DOCSIS: Data Over Cable Service Interface Specification
FRM: Forwarding Rules Manager
GBP: Group Based Policy
LISP: Locator/Identifier Separation Protocol
OVSDB: Open vSwitch DataBase Protocol
PCEP: Path Computation Element Communication Protocol
PCMM: Packet Cable MultiMedia
Plugin2OC: Plugin To OpenContrail
SDNI: SDN Interface (Cross-Controller Federation)
SFC: Service Function Chaining
SNBI: Secure Network Bootstrapping Infrastructure
SNMP: Simple Network Management Protocol
TTP: Table Type Patterns
VTN: Virtual Tenant Network
DLUX
VTN Coordinator
OpenStack Neutron
SDNI Wrapper
DDoS Protection
Network Applications Orchestrations & Services
AAA - AuthN Filter
OpenDaylight APIs (REST)
Base Network Service Functions
Topology Manager
Stats Manager
Switch Manager
FRM
Host Tracker
OpenStack Service
GBP Service
SFC
AAA
DOCSIS Abstraction
VTN Manager
OVSDB Neutron
Plugin2OC
LISP Service
L2 Switch
SNBI Service
SDNI Aggregator
Controller Platform
Service Abstraction Layer (SAL)
(Plugin Manager, Capability Abstractions, Flow Programming, Inventory, etc.)
GBP Renderers
OpenFlow
1.0
1.3
TTP
OVSDB
NETCONF
PCMM/COPS
SNBI
LISP
BGP
PCEP
SNMP
Plugin2OC
Southbound Interfaces & Protocol Plugins
OpenFlow Enabled Devices
Open vSwitches
Additional Virtual & Physical Devices
Data Plane Elements (Virtual Switches, Physical Device Interfaces)

*Source:* Khattak et al. (2014)

**Table 2** SDN controller comparison table

| | *Open source* | *First release* | *Language support* | *Platform support* | *Activity* | *Rest API* | *Documentation* | *GUI* | *Open flow version* | *Clustered development* | *Open stack* |
|---|---|---|---|---|---|---|---|---|---|---|---|
| NOX | Yes | 2008 | C/C++ | Linux | Low | No | Low | No | 1.0 | Yes | No |
| Beacon | Yes | 2010 | Java | Linux | Low | No | Med | Yes | 1.0 | Yes | No |
| Trenta | Yes | 2011 | C/C++ | Linux | Med | No | Med | No | 1.0 | Yes | Yes |
| Pox | Yes | 2012 | Python | Linux | Low | No | Low | Yes | 1.0 | No | No |
| Open ML | Yes | 2012 | C/C++ | Linux | Low | No | High | Yes | 1.0, 1.3, 1.4 | Yes | Yes |
| Ryu | Yes | 2012 | Python | Linux | Med | No | Med | Yes | 1.0, 1.2, 1.3, 1.4, 1.5 | Yes | Yes |
| Floodlight | Yes | 2012 | Java | Linux | Med | Yes | High | Yes | 1.0 | Yes | Yes |
| Open daylight | Yes | 2013 | Linux MAC | Linux | High | Yes | High | Yes | 1.0, 1.2, 1.3 | Yes | Yes |
| Open contrail | Yes | 2013 | Linux | Linux | High | Yes | High | Yes | No support | Yes | Yes |
| ONOS | Yes | 2014 | Linux | Linux | High | Yes | Low | Yes | 1.0, 1.2, 1.3, | Yes | Yes |

*Source*: Salman et al. (2016)

### *2.3.5 Opendaylight controller*

Opendaylight controller is an open source-based project run by the Linux Foundation and is a module open platform for customising and automating networks of all sizes and sizes. It is developed with a focus on network programming and is designed based on commercial solutions that handle a variety of use cases in existing network environments. Figure 4 (Khattak et al., 2014) shows the structure of the Opendaylight controller 'HELIUM' version. It has excellent visibility and control and can dynamically use the network depending on the state, so it is optimised for traffic, topology, and equipment in a near real-time state. Cloud infrastructure in an enterprise or service provider environment is also highly capable of supporting a wide range of use cases. Table 2 (Salman et al., 2016) shows comparisons of SDN controllers.

### *2.4 Mininet and Mininet-WiFi*

Mininet is a wired network emulator. Mininet is a network emulator that runs a collection of end hosts, switches, routers, and links in a single Linux kernel. Operating system virtualisation capabilities and processes can be used to scale up to hundreds of nodes. Users can implement new network features or new architectures, test large-scale topologies with application traffic, and then deploy the same code and test scripts to real-world networks. Mininet was created at Stanford University to use network technology as a tool for researching and teaching. Mininet provides visibility into virtual SDNs consisting of OpenFlow Controller, OpenFlow-enabled Ethernet, Switch, and Ethernet networks from multiple hosts connected to the switch. Mininet-WiFi is a wireless network emulator. Mininet-WiFi developers expanded the capabilities of Mininet, a traditional wired-based SDN emulator, by adding virtual WiFi stations and access points (APs) based on standard Linux wireless drivers and 802.11_hwsim wireless simulation drivers. We also support the addition of several wireless devices in the Mininet network scenario and add classes based on AP-related location and mobile station attributes evaluation. Mininet-WiFi extends the underlying Mininet code by adding or modifying classes and scripts, adds new features to Mininet-WiFi, and supports all common SDN emulation features in standard Mininet network emulators.

## 3 Implementation and performance assessment

### *3.1 Configuring a performance assessment environment*

A composition diagram for the proposed system for performance evaluation of this SDWN is shown in Figure 5. It installed VMware12 on its PC, operated the Ubuntu 14.04 LTS version as a virtual machine, and operated Mininet-WiFi within the Ubuntu 14.04 LTS version. Unlike conventional wired Mininet, a WiFi dongle was used to run Mininet-WiFi for implementation in wireless environments. After running Mininet-WiFi, the POX controller configures two stations and one AP to link the information to the Pox controller through instructions.

**Figure 5** Suggested system configuration (see online version for colours)

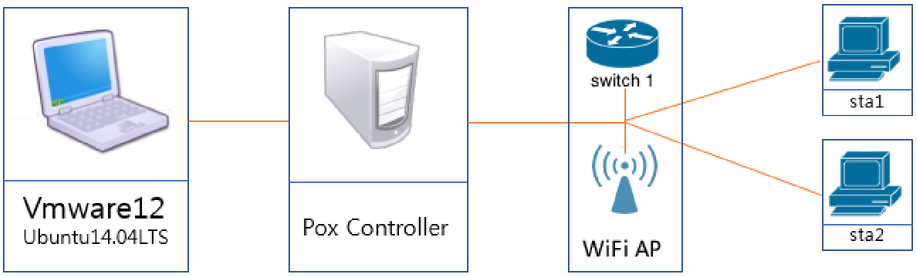
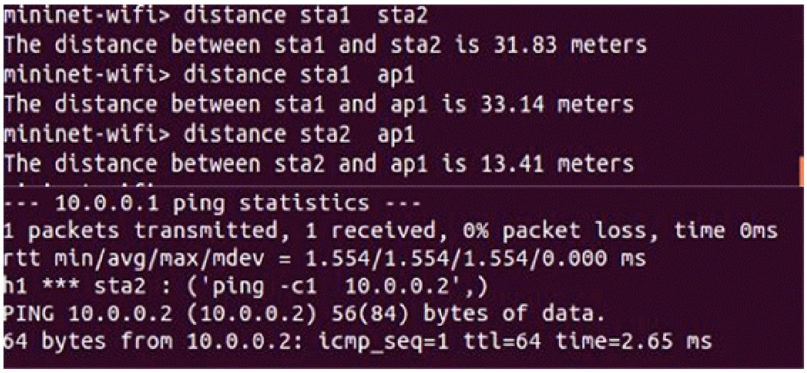
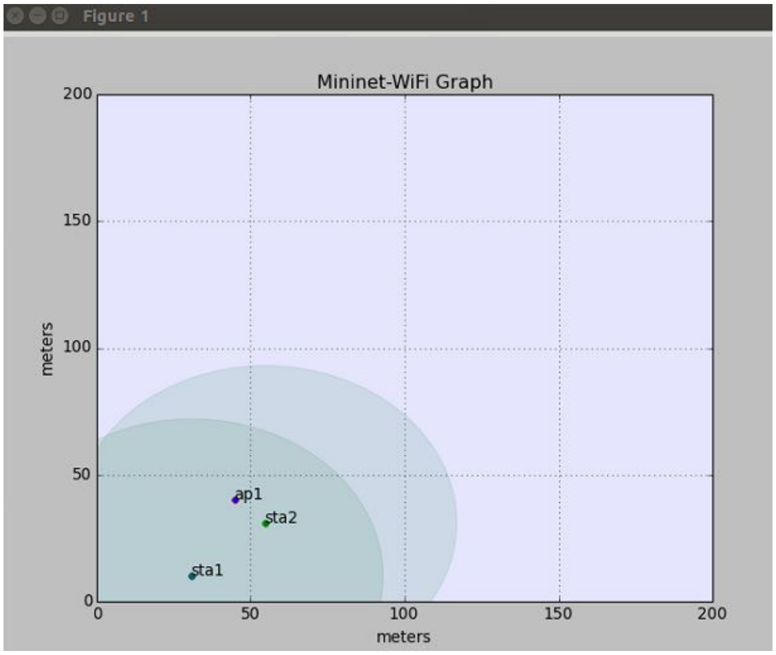


**Figure 6** Packet transmission time over distance (see online version for colours)



```
mininet-wifi> distance sta1  sta2
The distance between sta1 and sta2 is 31.83 meters
mininet-wifi> distance sta1  ap1
The distance between sta1 and ap1 is 33.14 meters
mininet-wifi> distance sta2  ap1
The distance between sta2 and ap1 is 13.41 meters
--- 10.0.0.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1.554/1.554/1.554/0.000 ms
h1 *** sta2 : ('ping -c1  10.0.0.2',)
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=2.65 ms
```

**Figure 7** Visualise station real-time mobility (see online version for colours)



### *3.2 Performance analysis of wireless SDN in mobility environment*

Figure 6 shows the packet transmission time according to the distance of the station. In Mininet network scenarios, classes based on the attribute evaluation of mobile stations, such as location and movement related to AP, are available as shown in Figure 7. Mininet-WiFi extends the basic Mininet code by adding or modifying classes or scripts and adds new features to Mininet-WiFi while supporting all common SDN emulation of standard Mininet network emulators, making it easy to configure topology through coding and performance evaluation, as shown in Figure 6. In addition, for highly mobile stations, packet transmission time can be determined by distance.

## 4 Conclusions

Mininet-WiFi is a wireless network emulator that supports virtual SDWN research. When sending packets to a highly mobile station, packet transmission time can be measured over distance and mobility can be verified in real-time. It can virtually check the mobility of stations through Mininet-WiFi without actual implementation, reducing the cost of implementation and making it easy for anyone to use because of its high availability. Connecting other high-performance controllers to Mininet-WiFi also facilitates streaming, home network setup, and management as well as packets. We adopted the Mininet emulator and analysed the performance of SDWN. The performance results revealed that the proposed work showed high performance in terms of transmission delays and resource usage. In the future, we aim to extend the SDWN by incorporating deep Q-learning techniques and developing intelligent SDWN. We intend to include various performance metrics such as throughput, bandwidth utilisation, congestion, latency, etc. and provide a detailed comparison with the state-of-the-art existing schemes.

## Acknowledgements

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korean Government (MSIT) (No. 2022R1A2C1003549).

## References

Asadollahi, S. and Goswami, B. (2017) 'Experimenting with scalability of floodlight controller in software defined networks', *International Conference on Electrical, Electronics, Communication, Computer and Optimization Techniques*.

Casalicchio, E. and Silvestri, L. (2011) 'Architectures for autonomic service management in cloud-based systems', *2011 IEEE Symposium on Computers and Communications*, pp.161–166.

Erickson, D. (2013) 'The beacon OpenFlow controller', *2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw.*, pp.13–18.

Fontes, R.R., Afzal, S., Brito, S., Santos, M. and Rothenberg, C.E. (2015) 'Mininet-WiFi: emulating software-defined wireless networks', *IEEE in CNSM*, Barcelona, Spain.

Ghaffar, Z. et al. (2021) 'A topical review on machine learning, software defined networking, internet of things applications: research limitations and challenges', *Electronics*, Vol. 10, No. 8, p.880.

Indrawan, G., Gunadi, I. and Sandhiyasa, I. (2022) 'REST API and real-time notification of SIsKA-NG mobile for the academic progress information system', *Information and Communication Technology for Competitive Strategies (ICTCS 2020)*, Springer, Singapore, pp.193–201.

Jamil, F. and Kim, D. (2021) 'An ensemble of a prediction and learning mechanism for improving accuracy of anomaly detection in network intrusion environments', *Sustainability*, Vol. 13, No. 18, p.10057.

Khattak, Z.K., Awais, M. and Iqbal, A. (2014) 'Performance evaluation of OpenDaylight SDN controller', *2014 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, pp.671–676, doi: 10.1109/PADSW.2014.7097868.

*Lab 08: Multi-Tenant Data Center Network – Part I* (n.d.) cs42200:spring19:labs:lab08 [online] https://courses.cs.purdue.edu/cs42200:spring19:labs:lab08 (accessed 7 May 2022).

Persia, S. et al. (2021) 'Ns3 and Mininet codes to investigate complete 5G networks', *2021 AEIT International Annual Conference (AEIT)*, pp.1–6, DOI: 10.23919/AEIT53387.2021.9626924.

Salman, O., Elhajj, I.H., Kayssi, A. and Chehab, A. (2016) 'SDN controllers: a comparative study', *18th Mediterranean Electrotechnical Conference*.

Sameer, M. and Goswami, B. (2018) 'Experimenting with ONOS scalability on software defined network', *Journal of Advanced Research in Dynamical and Control Systems*, Vol. 10, 14-Special Issue, pp.1820–1830.

Siddiqui, S. et al. (2022) 'Toward software-defined networking-based IoT frameworks: a systematic literature review, taxonomy, open challenges and prospects', *IEEE Access*, Vol. 10, pp.70850–70901, DOI: 10.1109/ACCESS.2022.3188311.

Xiang, Z., Pandi, S., Cabrera, J., Granelli, F., Seeling, P. and Fitzek, F.H.P. (2021) 'An open source testbed for virtualized communication networks', *IEEE Communications Magazine*, February, Vol. 59, No. 2, pp.77–83, DOI: 10.1109/MCOM.001.2000578.

Yan, B. et al. (2022) 'Flowlet-level multipath routing based on graph neural network in OpenFlow-based SDN', *Future Generation Computer Systems*, Vol. 134, pp.140–153.

Younus, M.U. et al. (2019) 'A survey on software defined networking enabled smart buildings: Architecture, challenges and use cases', *Journal of Network and Computer Applications*, Vol. 137, pp.62–77.