Development Environment for Software-defined Smart City Networks

Filip Holik[0000−0001−6595−0419]

Norwegian University of Science and Technology, Teknologivegen 22, 2815 Gjøvik, Norway
filip.holik@ntnu.no

Abstract. Smart cities are becoming the key technology as more and more people are attracted to living in urban environments. Increased number of inhabitants and current issues such as pandemic restrictions, terrorist threats, global warming and limited resources require maximum effectivity in all areas of city operations. To support these dynamic demands, software-based approaches for network management are being used. They provide programmability, which allows implementation of almost any functionality, but is dependent on quality of the developed application. To ensure the best possible quality, developers need an environment on which they can develop and test the application effectively. Such an environment should be as close to a real network as possible, but as easy to use as an emulated network.
This paper describes a method for creating a flexible and inexpensive practical environment for developing and testing applications for software-defined smart city networks. The paper analyzes four relevant open source controllers, compares their features and suitability for smart city applications; and provides a guideline for creating inexpensive SDN capable switches from general purpose single board Raspberry Pi computers. The presented environment can be deployed easily, but its hardware nature allows real performance measurements and utilization of IoT-based nodes and sensors. This is verified in an use case smart city topology.

Keywords: SDN controllers · Smart city · Software-defined networks · Software switch · Raspberry Pi.

1 Introduction

Smart city is a concept for enhancing current mega cities by innovative technologies from information and communication fields with the goal of improving every aspect of the city's operations. This includes economy, transportation, safety, waste and resources managements, and many others. The smart city market is steadily growing with investments increasing about 20% every year and expected to reach almost 200 billion U.S. dollars in 2023 [26]. Recent pandemic events and their negative impact on the economy might lead to an assumption that these expanses will be reduced, but the opposite is more likely true. Smart cities will be more important than ever, as their functionalities might significantly help with enforcing quarantine restrictions, support contact tracing, and check social distancing. The increased effectiveness of all city's operations plays an even greater role in these demanding situations.

Growth of smart cities puts requirements on corresponding underlying technologies, especially on communication networks. These networks must cope with stringent requirements on performance, reliability, scalability and security, which can change significantly based on dynamically fluctuating city needs. From this perspective, the static concept of traditional networking is ineffective and outdated. Only innovative software-based approaches such as software-defined networks (SDN) can cope with these dynamic conditions effectively.

SDN is a network concept of physical separation of forwarding and control layers on a networking device. While the forwarding layer is left on the device, the control layer is placed on a centralized element called the SDN controller. The controller provides management of the entire network and can be extended with custom made applications - for example to fit specifically smart city scenarios.

Functionality of SDN is based on the quality of its control application. Development of these applications is a demanding task, especially in complex and large scale networks such as smart cities. These networks are spread across vast areas and must remain fully functional, which eliminates the possibility of predeployment testing. SDN applications must therefore be developed and tested on dedicated, often only emulated networks. While this approach is quick and simple, it does not provide practical insight into the network operations. On the other hand, use of real networking devices only for development is expensive, time consuming and inflexible. The approach described in this paper combines the advantages of both approaches.

2 Related Work

SDN is widely accepted as a suitable technology for demanding smart city networks and a lot of research work has been done in this area [5–8, 11, 16, 18, 25, 28]. Authors in [8] summarized the main advantages of SDN in smart cities: intelligence, scalability and integration; and limitations of traditional networks: users identification based on IP addresses, problematic security and ineffective mobility management. Further research proved that SDN is capable of handling real-time applications, including in problematic areas such as wireless sensor networks [7, 11].

Activities towards real world deployment and practical verification are emerging. A large scale integration of SDN, cloud and IoT devices was tested on 8 million inhabitants Guadalajara smart city [5]. An emulated multi-tenant network corresponding to a virtual smart city of Poznan, integrated metro scale IoT network, cloud orchestrator and the SDN controller in [18].

Most of the research work in the area of SDN controllers comparison is focused either on security [2, 27] or performance [21, 22, 29, 30]. The most relevant comparison of Rosemary, Ryu, OpenDaylight and ONOS summarized basic features of these controllers, but the main focus of the work was on security comparison using the STRIDE (Spoof, Tamper, Repudiate, Information Disclose, DoS and Elevate) model [2]. Software reliability of 10 different versions of the ONOS controller was analyzed in [27]. Performance comparison work targeted mostly outdated or experimental controllers [21, 22, 30]. The most advanced open source controllers - OpenDaylight and ONOS - were compared only in [29], where authors performed tests in 5 different scenarios. In most of these scenarios, the ONOS controller achieved slightly higher performance.

The idea of creating an SDN-enabled switch from a single board computer by installation of a software switch is not new and was first researched in [15]. Authors installed Open vSwitch (2.0.90), which supported OpenFlow 1.0, into the first model of Raspberry Pi and tested its performance. Despite this software and hardware, the measured performance was comparable with 1 Gbps net-FPGA (Field Programmable Gate Arrays), which costs approximately 30 times as much as the Raspberry Pi. The following paper [12] analyzed a stack of four Raspberry Pi devices controlled by the ONOS controller. This work was followed by several other papers [1, 3, 4, 14] implementing more recent versions of Open vSwitch and using newly emerging Raspberry Pi models. None of the mentioned work considered use of the created environment for practical development of SDN applications for scenarios such as smart cities. The closest work in this area is providing QoS in IoT networks on 4-port switches made from Raspberry Pi 3 devices [17].

3 Open Source SDN Controllers

The key component of a software-defined network is a controller, which manages all connected devices, provides networking functions, collects traffic statistics and has interfaces for remote control and advanced applications integration. SDN controllers can be classified into open source and commercial. This section describes features of the four most relevant open source controllers for smart city scenarios.

3.1 Ryu

Ryu [24] is one of the simpler SDN controllers and has only basic functions. It is written in Python and uses module structure for various networking functionalities. These modules are placed in separated Python files and their use has to be specified during each controller launch via the *ryu-manager* command. The controller has extensive documentation, which contains examples of code implementation and format of OpenFlow messages in JSON. The community also provides a freely accessible Ryu book [23], which explains several modules and describes process of developing custom applications.

The controller is ideal for anyone starting with general SDN development. It has a relatively shallow learning curve and the strict module separation allows safe and quick development of custom functionality. It is also suitable for quick establishment of network connectivity and for testing specific functions.

In the area of smart cities, the controller is missing advanced features and it is therefore not recommended for these deployments. Applications developed for this controller would have to be migrated into more suitable controllers before the real deployment.

3.2 Floodlight

Floodlight controller [9] provides a compromise between the simplest and most advanced controllers. It has clear documentation, straightforward installation, basic configuration, but also provides GUI and supports even advanced features including high availability. The controller is written in Java and uses similar modular architecture as more advanced controllers.

The controller provides a set of extensible Representational State Transfer (REST) APIs and the notification event system. The APIs can be used by external applications to get and set the state of the controller, and to allow modules to subscribe to events triggered by the controller using the Java Event Listener.

Floodlight supports basic features necessary for every smart city deployment - namely GUI and high availability. It does not have more advanced features such as security, support of legacy devices, or ability to dynamically adjust modules while running, but this fact is compensated by its relative simplicity and low hardware requirements. Unfortunately, the controller lacks in frequency of updates, which would address security and other issues.

The controller is ideal for new developers learning to work with SDN, but using a near-realistic environment. Developed applications can also be used in real scenarios. However, for development of more advanced commercial applications, which would be used in real smart city scenarios, use of more advanced controllers is recommended.

3.3 OpenDaylight

OpenDaylight [20] is the most widespread open source SDN controller and it is defined as "a modular open platform for customizing and automating networks of any size and scale". OpenDaylight is written in Java and it is being used as a base for many commercial controllers, which extend its functionality.

Use of OpenDaylight controller requires significantly more resources than previous controllers - in terms of hardware, knowledge, installation and initial configuration. Developing custom applications is even more demanding as the controller has a complex architecture. Moreover, the official documentation is not nearly as user friendly and complete as in previous controllers, and described features are often relevant only for older versions of the controller.

OpenDaylight is an ideal controller for real smart city scenarios. It can reliably monitor and control large scale networks with a high number of connected nodes. The controller is focused on security and reliability and allows configuration changes or installation of a new functionality without a need for restarting the controller. The controller is also in active development and new major versions are regularly released every 6 months with minor updates available as needed. The fact that it is widely used as a base for commercial controllers proves its suitability for real world use.

3.4 Open Networking Operating System

Open Networking Operating System (ONOS) [19] is a similar controller to OpenDaylight, but its main focus is on resiliency, scalability and deployment in production environments.

ONOS provides a slightly more detailed documentation than OpenDaylight, but it is still not so well structured and complete as in the case of Ryu or Floodlight. Use of the controller requires similar resources as in the case of OpenDaylight.

ONOS is the second analyzed controller, which is ideal for real smart city deployments. It is similar to OpenDaylight, but offers several unique features. It is slightly more oriented towards resiliency and it is the only controller which supports individual removal of functionalities even during the controller operations. Its functionalities can be also installed and activated without a need of restarting the controller. New versions are being released in approximately 3 month intervals with incremental updates available if needed.

3.5 Summary of Controllers Features

Table 1 summarizes supported features of each of the analyzed controllers. It includes only features which are officially supplied with the controller. Other features provided by independent developers and communities can be additionally installed.

4 Custom SDN-enabled Switches

The second key component of SDN are forwarding devices, which are connected to the controller. They are being called switches, although they support all ISO/OSI layers and not only layer 2 forwarding. They can have the following forms:

1. Traditional switch with optional OpenFlow support (limited features)
2. Software switch (slow performance)
3. Specifically developed OpenFlow device (full features)

Hardware networking devices supporting the OpenFlow protocol are relatively expensive and their use only for development purposes might not be economically sustainable. Use of software devices is much more efficient method, but it does not allow practical verification and native connection of specific hardware IoT sensors and nodes. A solution using advantages of both approaches is to create an SDN-enabled device from a cheap generic single board computer with an integrated software switch.

Table 1. Features of Compared SDN Controllers

Functionality / Controller	***RYU***	***FLT***	***ODL***	***ONOS***
Basic functionality (L2, L3, STP, VLANs, ACL, FW)	✓	✓	✓	✓
GUI (S = secure)	-	✓	S	S
Dynamic routing (M = MPLS)	-	-	✓	M
Virtualization, OpenStack	-	✓	✓	✓
Fault tolerance	-	✓	✓	✓
Quality of Service (QoS)	✓	✓	✓	-
L2 link aggregation	✓	-	✓	-
Intent networking	-	-	✓	✓
Service Function Chaining	-	-	✓	✓
YANG Management	-	-	✓	✓
Virtual Private Network (VPN)	-	-	✓	-
Dynamic Host Configuration Protocol (DHCP)	-	✓	-	✓
Load-balancing	-	✓	-	✓
ISP support	-	-	-	✓
CAPWAC	-	-	✓	-
Performance monitoring	-	✓	-	✓
Machine-to-machine communication	-	-	✓	-
Legacy device support	-	-	✓	✓
Device drivers	-	-	✓	✓
Controllers cooperation	-	-	✓	-

4.1 Required Components

To create a custom SDN-enabled device from a single board computer, two components are required:

1. Software switch - the most widespread being Open vSwitch (OVS). It is an open source multilayer software switch written in C language. It is flexible, universal and supports various management protocols including OpenFlow. OVS can be deployed either as a software switch (for example in virtualized data center environments) or in a hardware device.
2. Hardware computer - can have form of a single board computer based on the ARM (Advanced RISC Machine) architecture. The most widespread type of this computer is Raspberry Pi. It has several models, which differs in size, performance and connectivity options as summarized in Table 2.

4.2 Installation

There are two methods of installation of Open vSwitch on the Raspberry devices default operating system - Raspbian:

1. Installation from Raspbian repository - it is the easiest method of installation as it requires only a single command: *sudo apt-get install openvswitch-switch.*

Table 2. Comparison of Raspberry Pi Models

Model	*Release Date*	*Price (USD)*	*CPU (GHz)*	*RAM (MB)*	*Ports*	*USB*
B	02/2012	25	1x0.7	512	1FE	2
A+	11/2014	20	1x0.7	256	-	1
B+	07/2014	25	1x0.7	512	1FE	4
Zero	11/2015	5	1x0.7	512	-	0
2B	02/2015	35	4x0.9	1024	1FE	4
3B	02/2016	35	4x1.2	1024	1FE	4
3B+	03/2018	35	4x1.4	1024	1GbE	4
3A+	11/2018	25	4x1.4	512	-	1
4B	01/2019	35/45/55	4x1.5	1024/2048/4096	1GbE	4
4B	05/2020	75	4x1.5	8192	1GbE	4

The main disadvantage of this method is that the repository might not include the most recent version of OVS (at the time of writing this paper, only version 2.3, which supports only OpenFlow 1.3 and older, was available).

2. Installation with Debian packages - the most up to date source code can be downloaded from Github [10]. This version already supports OpenFlow 1.4. The following commands show the installation. The last component will also perform initial configuration of the switch and sets it to automatic startup upon the system's boot.

```
# 1. Download the source code
git clone https://github.com/openvswitch/ovs.git
# 2. Compilation (requires build-essential and fakeroot tools)
#    X = the number of threads, which the compilation can use
DEB_BUILD_OPTIONS='parallel=X' fakeroot debian/rules binary
# 3. Installation of Dynamic Kernel Module Support (DKMS)
sudo apt-get install dkms
# 4. Compiled kernel package installation
sudo dpkg -i openvswitch-datapath-dkms_2.5.2-1_all.deb
# 5. Installation of a package for generation of unique IDs
sudo apt-get install uuid-runtime
# 6. Order-dependent installation of user-space packages
sudo dpkg -i openvswitch-common_2.5.2-1_armhf.deb
sudo dpkg -i openvswitch-switch_2.5.2-1_armhf.deb
```

4.3 SDN Configuration

Installed and configured OVS can be integrated with SDN. This requires establishing communication between the controller and the device. Two modes of this communication are available:

1. Unsecured communication - the basic form of connection uses standard TCP and can be setup with the following commands.

```
# 1. Create a virtual switch with a name S-NAME
sudo ovs-vsctl add-br S-NAME
# 2. Configure IP and TCP port (default 6653) of the controller
sudo ovs-vsctl set-controller S-NAME tcp:IP:PORT
# 3. Create a virtual interface in the /etc/dhcpd.conf
interface S-NAME
static ip_address = IP/PREFIX
# 4. Apply the configuration (or reboot the device)
sudo /etc/init.d/dhcpcd reload
# 5. Assign the physical interface to the virtual switch
sudo ovs-vsctl add-port S-NAME INT-NAME
```

2. Secured communication - this form uses TLS for encryption and it requires use of private keys and certificates. The required files can be generated by the OpenFlow public key infrastructure management utility, which can be managed by the *ovs-pki* command. Files generated on the device then have to be transferred to the controller. The procedure of how to load these files will vary based on the controller.

```
# 1. Create certification authorities
sudo ovs-pki init
# 2. Create private key and certificate for the controller
sudo ovs-pki req+sign C-NAME controller
# 3. Create private key and certificate for the switch
sudo ovs-pki req+sign S-NAME switch
# 4. Set the required files for the TLS configuration
sudo ovs-vsctl set-ssl
/home/S-NAME-privkey.pem
/home/S-NAME-cert.pem
/home/controllerca/cacert.pem
# 5. Enable TLS
sudo ovs-vsctl set-controller S-NAME ssl:IP:PORT
```

5 Use Case Verification

Installation and configuration of switches and controllers were verified on a topology simulating a small smart city as shown in Figure 1. Three Raspberry Pi 3B devices were used as SDN-enabled switches. The same platform cannot be used to host SDN controllers due to the different architecture (ARM vs x86-64) and low CPU performance. To make the use case environment as efficient as possible, a NUC8i7BEH mini-PC [13] was used for the SDN controller. Such a device is relatively cheap, has sufficient performance and low energy consumption. It is therefore ideal for this role.

5.1 Controllers Analysis

Four controllers from Section 3 were installed into separate virtual machines in order to test their performance - especially various RAM configurations. The

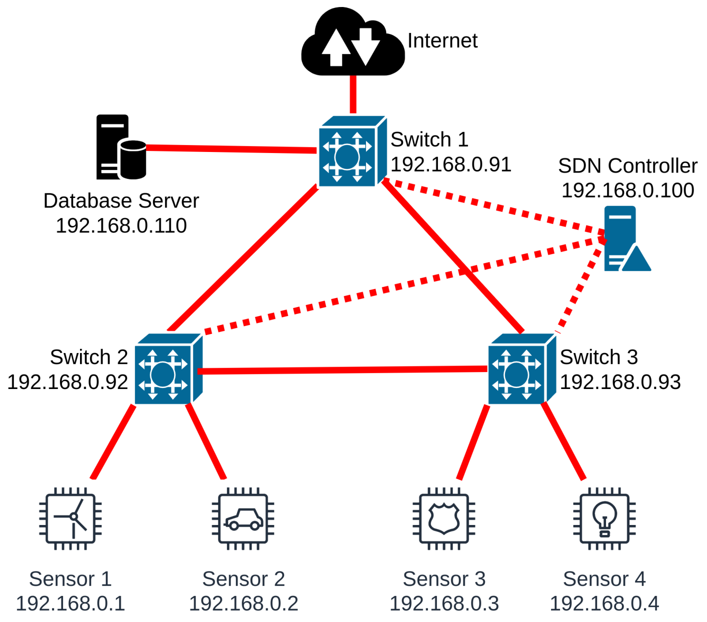

Fig. 1. Use case verification topology

main motivation was to determine the minimal amount for stable run of the controller. Results are presented in Table 3 together with approximate startup times and controllers support to launch modules.

Table 3. Controllers Performance Analysis

Controller	***Minimum RAM***	***Startup time****	***Support of modules launch***
Ryu	256 MiB	<30 seconds	At start (manually)
Floodlight	256 MiB	<20 seconds	At start (configuration file)
OpenDaylight	4096 MiB	<20 seconds	Dynamic start at run
ONOS	4096 MiB	<20 seconds	Dynamic start/stop at run

* Measured time is just approximate as it is highly dependent on the network topology size, the controller performance and its current load.

Results show that Ryu and Floodlight have very low memory requirements and can run on practically any device. On the other hand, OpenDaylight and ONOS require a device with at least 4 GiB of RAM even in the smallest network topologies. Startup times do not vary significantly between the controllers and should not play a role in the controller selection process.

In environments where the network should correspond to real smart cities, only OpenDaylight and ONOS are recommended as they are the only ones allowing to start new modules while the controller is running (and in the case of ONOS also stop and remove them).

5.2 Deployment Findings

The verification revealed the need to use specific features, which are summarized below to make any future deployment testing more effective.

1. Port numbers - the OVS device uses integer labeling. A specific port can be found with the following command:
```
    sudo ovs-ofctl dump-ports DEVICE-NAME INTERFACE-NAME
```
2. Datapath ID - is an identification number, which the controller uses to recognize connected devices. By default, the device's MAC address of the interface leading to the controller is used. This address can be configured with the following command (sets the MAC address to 1):
```
    sudo ovs-vsctl set bridge DEVICE-NAME
       other-config:hwaddr=00:00:00:00:00:01
```
3. Time synchronization - encrypted communication and use of certificates require synchronized time between the controller and devices. In this case, it is necessary to ensure the time synchronization, for example by the Precision Time Protocol (PTP). A time difference can lead to the following error:
```
    SSLError: [SSL: SSLV3_ALERT_BAD_CERTIFICATE]
```

6 Conclusions

The paper described issues of developing SDN applications for smart city scenarios. In two main sections, the topic of creating a practical and cost-effective environment for these scenarios was researched. Presented information was verified and tested on an use case topology of a small scale smart city network.

The analysis of four open source controllers summarized their key features and included recommendations for the most effective utilization of each controller in smart city networks. Ryu controller was recommended only for learning purposes and quick verification of connectivity as its deployment in smart cities is not feasible due to lack of features. Floodlight controller was identified as a compromise between simple Ryu and more advanced controllers. While it can be used for development and testing of smart city applications, its usage in real networks was also not recommended.

OpenDaylight and ONOS were identified as similarly advanced open source controllers. They require significantly more effort to deploy and manage, but because of supported features, they can be safely used in real world smart city networks. The final choice from these two controllers depends on the target application. OpenDaylight supports more functionalities while ONOS offers slightly higher performance and is more flexible in terms of controlling running features.

The presented approach for creating an SDN-enabled switch from a Raspberry Pi device can be used to quickly create an environment suitable for practical development testing not limited only to smart city scenarios.

References

1. Alipio, M., Udarbe, G., Medina, N., Balba, M.: Demonstration of Quality of Service mechanism in an OpenFlow testbed. pp. 443–447 (2016). https://doi.org/10.1109/IMCEC.2016.7867251
2. Arbettu, R., Khondoker, R., Bayarou, K., Weber, F.: Security analysis of OpenDaylight, ONOS, Rosemary and Ryu SDN controllers. pp. 37–44 (2016). https://doi.org/10.1109/NETWKS.2016.7751150
3. Ariman, M., Seçinti, G., Erel, M., Canberk, B.: Software defined wireless network testbed using Raspberry Pi of switches with routing add-on. In: 2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN). pp. 20–21 (2015)
4. Babayigit, B., Karakaya, S., Ulu, B.: An implementation of software defined network with Raspberry Pi. In: 2018 26th Signal Processing and Communications Applications Conference (SIU). pp. 1–4 (2018)
5. Cedillo-Elias, E., Orizaga Trejo, J.A., Larios-Rosillo, V., Arellano, L.: Smart Government infrastructure based in SDN Networks: the case of Guadalajara Metropolitan Area. pp. 1–4 (2018). https://doi.org/10.1109/ISC2.2018.8656801
6. Chakrabarty, S., Engels, D.W.: A secure IoT architecture for Smart Cities. In: 2016 13th IEEE Annual Consumer Communications Networking Conference (CCNC). pp. 812–813 (2016)
7. Din, S., Rathore, M.M., Ahmad, A., Paul, A., Khan, M.: SDIoT: Software Defined Internet of Thing to Analyze Big Data in Smart Cities. In: 2017 IEEE 42nd Conference on Local Computer Networks Workshops (LCN Workshops). pp. 175–182 (2017)
8. Fekih, A., Gaied, S., Yousef, H.: A comparative study of content-centric and software defined networks in smart cities. In: 2017 International Conference on Smart, Monitored and Controlled Cities (SM2C). pp. 147–151 (2017)
9. Floodlight community: Floodlight controller (2018), https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/overview
10. GitHub: Open vSwitch (2020), https://github.com/openvswitch/ovs
11. Hakiri, A., Gokhale, A.: Work-in-progress: Towards real-time smart city communications using software defined wireless mesh networking. In: 2018 IEEE Real-Time Systems Symposium (RTSS). pp. 177–180 (2018)
12. Han, S., Lee, S.: Implementing SDN and network-hypervisor based programmable network using Pi stack switch. In: 2015 International Conference on Information and Communication Technology Convergence (ICTC). pp. 579–581 (2015)
13. Intel: Intel® NUC Kit NUC8i7BEH with 8th Generation Intel® Core™ Processors (2021), https://www.intel.com/content/www/us/en/products/sku/126140/intel-nuc-kit-nuc8i7beh/specifications.html
14. Jaramillo, A.C., Alcivar, R., Pesantez, J., Ponguillo, R.: Cost Effective test-bed for Comparison of SDN Network and Traditional Network. In: 2018 IEEE 37th International Performance Computing and Communications Conference (IPCCC). pp. 1–2 (2018)
15. Kim, H., Kim, J., Ko, Y.B.: Developing a cost-effective OpenFlow testbed for small-scale Software Defined Networking. pp. 758–761 (2014). https://doi.org/10.1109/ICACT.2014.6779064
16. Munir, M.S., Abedin, S.F., Alam, M.G.R., Tran, N.H., Hong, C.S.: Intelligent service fulfillment for software defined networks in smart city. In: 2018 International Conference on Information Networking (ICOIN). pp. 516–521 (2018)

17. Nguyen, Q.H., Ha Do, N., Le, H.: Development of a QoS Provisioning Capable Cost-Effective SDN-based Switch for IoT Communication. In: 2018 International Conference on Advanced Technologies for Communications (ATC). pp. 220–225 (2018)
18. Ogrodowczyk, L., Belter, B., LeClerc, M.: IoT Ecosystem over Programmable SDN Infrastructure for Smart City Applications. In: 2016 Fifth European Workshop on Software-Defined Networks (EWSDN). pp. 49–51 (2016)
19. Open Networking Foundation: Open Network Operating System (2020), https://www.opennetworking.org/onos/
20. OpenDaylight Project: OpenDaylight (2018), https://www.opendaylight.org/
21. Priyadarsini, M., Bera, P., Bampal, R.: Performance analysis of software defined network controller architecture—a simulation based survey. pp. 1929–1935 (2017). https://doi.org/10.1109/WiSPNET.2017.8300097
22. Rastogi, A., Bais, A.: Comparative analysis of software defined networking (SDN) controllers — In terms of traffic handling capabilities. pp. 1–6 (2016). https://doi.org/10.1109/INMIC.2016.7840116
23. RYU project team: RYU SDN Framework (2016), https://osrg.github.io/ryu-book/en/Ryubook.pdf
24. Ryu SDN Framework Community: Ryu SDN Framework (2017), https://osrg.github.io/ryu/
25. Saqib, M., Khan, F.Z., Ahmed, M., Mehmood, R.M.: A critical review on security approaches to software-defined wireless sensor networking. International Journal of Distributed Sensor Networks **15**(12), 1550147719889906 (2019). https://doi.org/10.1177/1550147719889906
26. Statista: Technology spending into smart city initiatives worldwide from 2018 to 2023 (2020), https://www.statista.com/statistics/884092/worldwide-spending-smart-city-initiatives/
27. Vizarreta, P. et al.: An empirical study of software reliability in SDN controllers. pp. 1–9 (2017). https://doi.org/10.23919/CNSM.2017.8256002
28. Wang, S., Gomez, K.M., Sithamparanathan, K., Zanna, P.: Software Defined Network Security Framework for IoT based Smart Home and City Applications. In: 2019 13th International Conference on Signal Processing and Communication Systems (ICSPCS). pp. 1–8 (2019)
29. Yamei, F., Qing, L., Qi, H.: Research and comparative analysis of performance test on SDN controller. pp. 207–210 (2016). https://doi.org/10.1109/CCI.2016.7778909
30. Zhao, Y., Iannone, L., Riguidel, M.: On the performance of SDN controllers: A reality check. In: 2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN). pp. 79–85 (2015)