An SDN Architecture for IoT Networks Using ONOS Controller

Calin-Marian Iurian
Communications Department
Technical University of Cluj-Napoca
Cluj-Napoca, Romania
Marian.Iurian@com.utcluj.ro

Iustin-Alexandru Ivanciu
Communications Department
Technical University of Cluj-Napoca
Cluj-Napoca, Romania
Iustin.Ivanciu@com.utcluj.ro

Beniamin Mihai Marian
Communications Department
Technical University of Cluj-Napoca
Cluj-Napoca, Romania
beniaminm43@gmail.com

Daniel Zinca
Communications Department
Technical University of Cluj-Napoca
Cluj-Napoca, Romania
Daniel.Zinca@com.utcluj.ro

Virgil Dobrota
Communications Department
Technical University of Cluj-Napoca
Cluj-Napoca, Romania
Virgil.Dobrota@com.utcluj.ro

2020 19th RoEduNet Conference: Networking in Education and Research (RoEduNet) | 978-0-7381-1265-7/20/$31.00 ©2020 IEEE | DOI: 10.1109/ROEDUNET51892.2020.9324887

Abstract— In order to evaluate the ONOS controller for Software Defined Networks (SDN) in an Internet of Things (IoT) context, Mininet was installed and configured in a virtual machine. Two different scenarios were created: the first one tests the general functionality of the proposed architecture with the IoT data being sent from the MQTT.fx client to the Orange Live Objects platform. For the second scenario different flow rules were configured in order to send the time-critical IoT data from the Mosquitto client to the broker on the path with the lowest delay. The proposed solution can easily be extended to accommodate other types of IoT devices and platforms.

Keywords— Internet of Things; Mininet; MQTT; ONOS controller; Orange Live Objects; Software Defined Networking.

I. INTRODUCTION

The IoT technology emerged about 20 years ago and since its first appearance, the number of connected devices increased significantly. Devices in the Internet of Things are defined by the capability to transfer data over an Internet connection. Their functionality is based on the sensor capacity to collect and share information by connecting to an IoT gateway or other edge platform. From this point, the information is sent to be analyzed locally or to a cloud platform.

The authors of [1] predicted about 50 billion connected devices (actuators, machines, and sensors from all around the world) in 2020 and now this is reality. By 2030, this number is expected to rise to 125 billion, with every customer of IoT devices having around 15 connected devices. Another study [2] suggests that 23% of current large-scale IoT projects are smart cities and 40% of IoT devices are used in healthcare industry in 2020.

IoT can be brought into practice in many ways as it is a huge infrastructure of things, services and objects. For example, current IoT wireless technologies can be divided into the six main categories illustrated in Fig. 1 [3]. This wide variety on how the information is processed and transmitted also leads to numerous vulnerabilities and security issues as indicated in paper [3]. In terms of data transfer, the authors of [4] introduce a thorough comparison between the most popular IoT protocols. The results showed that MQTT has the best performance when it comes to data latency and reliability, thus making it a perfect candidate for any IoT system that requires low latency.

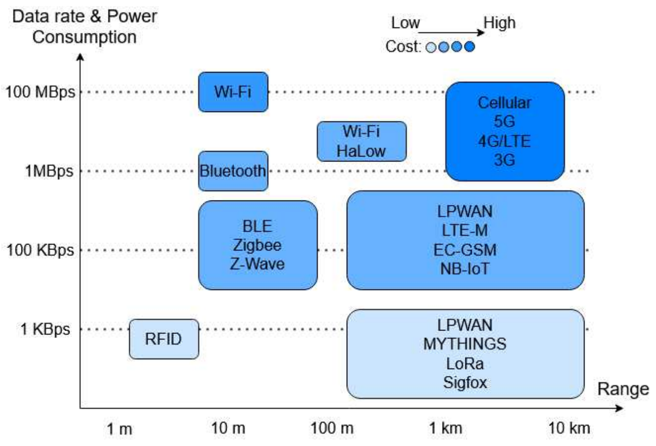


Fig. 1. IoT wireless availability

Data routing to an IoT gateway or an edge device is an essential part of any large-scale wireless network and smart solution. The ever-increasing number of applications will generate huge amounts of data. This in turn requires more and more attention for low-power devices operating in lossy networks. Therefore, energy-efficient routing of data and enhancement of intelligent routing protocols become critical to any long-term sustainable result. According to [5], an intelligent routing protocol has the capacity to harness the entire power of any heterogeneous, dynamic, and complex network that is defined by multiple dynamic transformations such as a real-time topology change. A complex study presented in [6] analyzes the most important features of routing in IoT and proposes several paths for future research, in both industrial and academic environments.

The fast growth of technologies, such as cloud computing, big data, or Internet of Things, also increases the complexity of the network deployments, so much so that legacy networks can no longer keep up with new challenges. One of the possible answers to these challenges is the introduction of a high level of abstraction and automation in the delivery and processing of IoT applications, namely by using SDN [7]. This separates the network control plane from the data plane. It relies on virtualization, in order to enable physical network devices to be replaced by specialized software. SDN aims to eliminate complex vendor configuration and complicated management by performing centralized control of the entire network.

A performance evaluation of the Open Network Operating System (ONOS) and Floodlight SDN controllers is presented in [8]. The results showed that in terms of transfer rate, jitter and delay, the ONOS controller performs better for both TCP and UDP traffic. Paper [9] explores the SDN/NFV (Network Functions Virtualization) architectures for deploying IoT gateways for multi-tenant services and devices. ONOS controller proved to be a good solution for deploying scalable networks. Surveys, such as [10] and [11], demonstrate that the high modularity and scalability of the ONOS controller make it highly suitable for application development. Considering the low latency and the support for different deployments and network architectures, the ONOS controller can be a proper solution for integrating both northbound and southbound interfaces into an IoT environment.

The main motivation for this paper was to take advantage of the characteristics of SDN and apply them to an IoT network. We wanted to test whether the ONOS controller is suitable for developing real applications that meet the requirements inherent to IoT networks: low latency, flexibility and scalability. The rest of the paper is organized as follows: Section II presents an overview of Software Defined-Networking and the ONOS controller. The next section describes the implementation of a network for IoT infrastructures, followed by preliminary results as proof of concept. The paper ends with conclusions and future work.

II. Overview of SDN and ONOS Controller

Legacy IP-based networks suffer from lack of flexibility and scalability required by most modern applications, especially in IoT. Software Defined Networking has emerged as a solution for powerful virtualization and centralization control, allowing the separation of data plane from the control plane. In the physical network, the forwarding tables are calculated by each router or switch. In the new architecture, the SDN controller manages flow control to the routers or switches using the southbound interface and applications via the northbound interface. The northbound Applications Programming Interfaces (APIs) communicate the application requirements so that the controller can establish the most appropriate path in terms of quality of service, security and management. The southbound interface interacts with the controller using several protocols, including OpenFlow, OpenvSwitch Data Base and different network machine configuration protocols [12]. Using a communication protocol, the controller forwards the packets across a system of interconnected switches.

The traditional SDN approach uses a single controller to manage the entire network. The architecture consists of three layers as shown in Fig. 2 [7]. The infrastructure layer represents the software infrastructure and the supporting physical hardware of the entire network, the SDN controller serves as the control plane, while the application management plane is responsible for the applications and services running on the network.

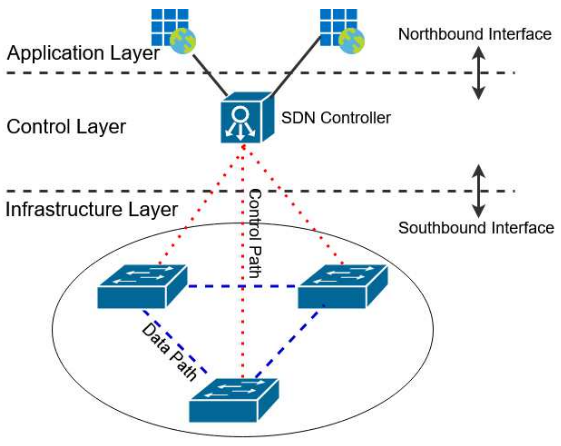


Fig. 2. SDN Architecture

ONOS is an open source, distributed controller designed for SDN architectures. Its main goal is to solve the scalability, availability, and performance issues in modern networks. ONOS is composed by an extensible and interchangeable platform and a set of applications. For the southbound interface, ONOS supports OpenFlow, Network Configuration Protocol (NETCONF), and Path Computation Element Protocol (PCEP). As northbound interface, it uses Representational State Transfer applications (REST APIs) [13]. This controller has a multi-level architecture divided into application, core and providers. It encapsulates multiple subsystems to support topology discovery and path calculations. The core transports the information to application layer for making the decision. Next, the OpenFlow protocol communicates the selected path to the switches/devices across the network. The southbound interface is responsible for discovering new devices added to the network and conveys the updated paths at upper layers of the ONOS stack [14]. Fig. 3 illustrates the logical flow of how ONOS controls the topology and dynamically computes the paths in an IoT network.

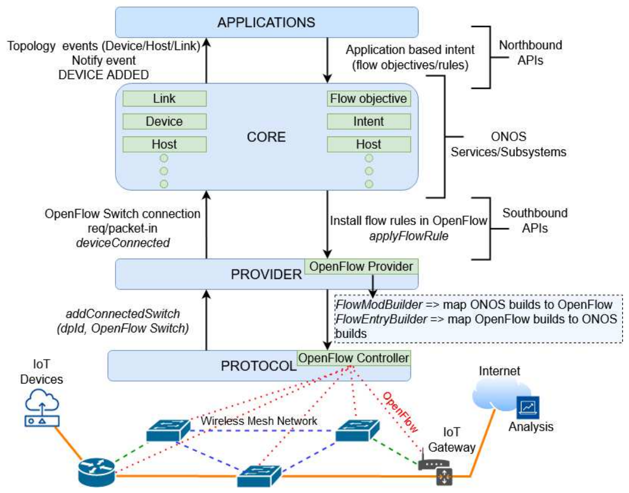


Fig. 3. Logical flow in ONOS

III. Implementation of an ONOS-based SDN Architecture

This paper presents the implementation of an IoT network topology, managed by the ONOS SDN controller. The proposed architecture, illustrated in Fig. 4 uses five OpenvSwitches as forwarding devices. The five hosts play different roles: (1) runs both a Mosquitto client and an MQTT.fx client. The IoT data generated in (1) is then sent to the (2) Mosquitto Broker for edge processing and to the (3) Internet Gateway for storing in the Cloud-based Orange Live Objects platform. Note that Host4 and Host5 are only used to generate cross-traffic.

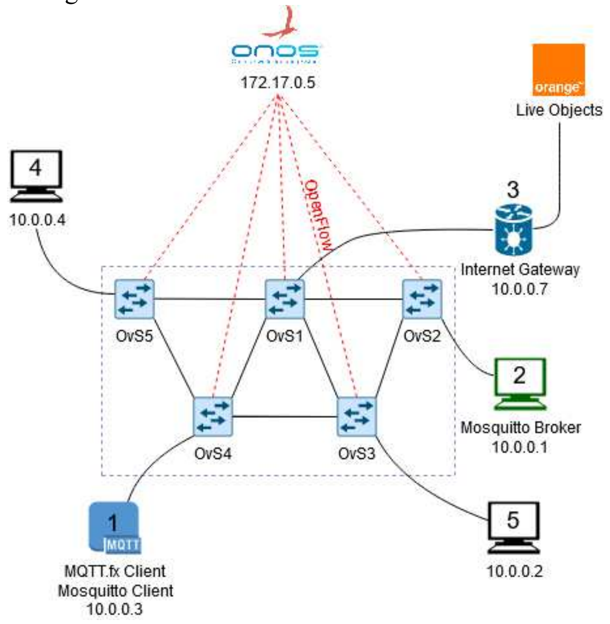


Fig. 4. Proposed SDN Architecture for IoT based on ONOS

The IoT infrastructure is developed in Ubuntu 18.04.5 LTS. The SDN virtual network was deployed in Mininet environment which is a great tool for designing and expanding virtual networks inside a real machine [15]. The custom topology was designed using a Python script.

A. Deployment of the ONOS Controller

Even if ONOS allows for a distributed control architecture, due to the lower complexity of the network, we chose to implement a single centralized controller to handle all the traffic. In order to install the ONOS controller on a single machine, first it must be downloaded from the ONOS website. Next, the `tar` command will extract the archive into /opt, and the directory is renamed to "onos". ONOS can be run directly calling its start-stop script, located in `/opt/onos/bin` directory. Fig. 5 illustrates the commands required for a complete installation:

```
sudo wget -c
http://downloads.onosproject.org/release/onos$ONOS
_VERSION.tar.gz
sudo tar xzf onos-$ONOS_VERSION.tar.gz
sudo mv onos-$ONOS_VERSION onos
/opt/onos/bin/./onos-service start
```

Fig. 5. ONOS installation commands

Once the SDN controller is up, we used the `sudo mn` command and the Python script to start the Mininet topology. After initialization, we checked the connectivity by performing a simple ping test from one host to another. As expected, the ping failed, because initially there are no flow rules installed on the data-plane, to forward the traffic properly. Moreover, the devices are not even discovered by ONOS. This can easily be fixed by using the default "Reactive Forwarding" application that installs forwarding flows on demand, before a sender starts the transmission [16]. The command `app activate org.onosproject.fwd` activates this application. Now the hosts can ping each other and all the devices in the topology are visible (Fig. 6) in the ONOS GUI which can be accessed from a web browser at: http://172.17.0.5:8181/onos/ui/index.html

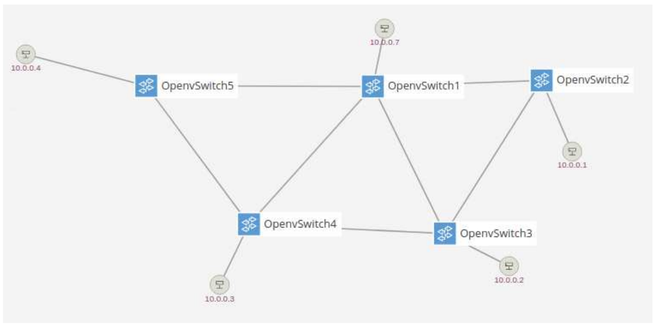


Fig. 6. Network topology in ONOS

B. Configuring MQTT Clients and Brokers

An IoT device needs to use a communication protocol in order to exchange data with a server from a distant location. MQTT is the standard publish/subscribe protocol for connecting smart devices in an IoT network. MQTT.fx is a popular MQTT client tool that connects with IoT Hub services (e.g., Orange Live Objects) to publish or subscribe to messages. In our scenario, MQTT.fx is used to simulate IoT sensors and actuators. In order to install the client on one of the Mininet hosts, the network needed an Internet connection to download the software. For that, we used the `--nat` command to deploy a NAT device that connects the topology in Mininet to the physical network. In order to test the Internet connectivity, we issued a ping command from Host4 to a public website. Fig. 7 illustrates the path used for this scenario and highlights the role of the newly added device, with the IP address 10.0.0.7, as an Internet Gateway. The values on the links represent the amount of network traffic.

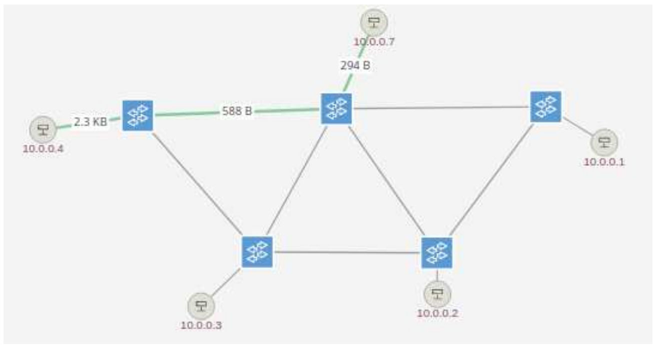
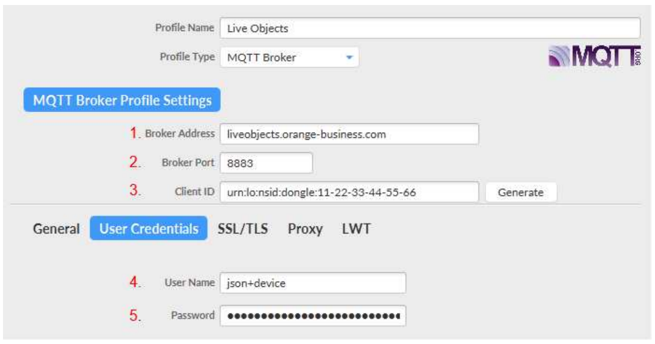


Fig. 7. Network connectivity to Internet

In our proposed architecture Orange Live Objects is used to retrieve the data from the MQTT.fx client. Orange Live Objects is a software suite for IoT/M2M integration offering a set of tools for device interconnection. It acts as a standard

MQTT message broker and can be used with or without encryption (TLS/ SSL layer). In order to authenticate on Live Objects in MQTT, we generated an API key used as a password for an MQTT.fx device connecting to the platform. It is important to have a direct connection to the Internet because a proxy can block the communication. We installed MQTT.fx on Host3 and configured five parameters required for the connection to Live Objects (Fig. 8). The first block (1) represents the broker's URL: `liveobjects.orange-business.com`. The port (2) used by the broker is the 8883 in SSL, or 1883 without SSL. The client ID (3) can be personalized but must respect the format of Live Objects (urn:lo:nsid:type). In the user credential part (4), we filled in the username or login of the connection that tells Live Objects the mode used. Since we simulate a device sending data to Live Objects, the format should be "json+device". The final parameter (5) asks for the authentication password which in this case is the API key previously generated from the Live Objects account. The green icon indicates that the connection was successful.



Profile Name: Live Objects
Profile Type: MQTT Broker

MQTT Broker Profile Settings

1. Broker Address: liveobjects.orange-business.com
2. Broker Port: 8883
3. Client ID: urn:lo:nsid:dongle:11-22-33-44-55-66 Generate

General | User Credentials | SSL/TLS | Proxy | LWT

4. User Name: json+device
5. Password: ••••••••••••••••••••••••••

Fig. 8. Connection dashboard in MQTT.fx

In the Live Objects "Devices" menu, the client ID `urn:lo:nsid:dongle:11-22-33-44-55-66` used with our object is marked with a green dot, meaning that the MQTT client is connected (Fig. 9).

Auto-created device (mqtt / urn:lo:nsid:dongler :11-22-33-44-55-6 6)	urn:lo:nsid:dongler :11-22-33-44-55-6 / 6	MQTT	● Online

Fig. 9. MQTT.fx – Live Objects connection

In order to highlight time-critical IoT scenarios, in which the data needs to be processed in the edge, we also installed a Mosquitto broker and a Mosquitto client; the two are lightweight implementations of the MQTT protocol, suitable for low-power sensors and embedded devices [18]. Unlike the MQTT.fx client which sends data to an external IoT broker, both Mosquitto entities are running locally. We configured Host2 as a broker while Host1 is the client.

IV. Experimental Results

Two scenarios were created in order to test the traffic forwarding functions of the ONOS controller. The first scenario involves the MQTT.fx client and the Orange Live Objects platform. In this case, we just wanted to demonstrate the general functionality of the proposed architecture and the ability of ONOS to redirect the traffic in case of a link failure. The second scenario refers to the Mosquitto client and broker. In this case, we simulated the transfer of critical IoT data which needs to be processed in the edge. As such, several forwarding rules were created to ensure the traffic followed the path with the lowest latency.

A. Scenario 1: MQTT.fx and Orange Live Objects

For the first scenario data generated by the MQTT.fx client is sent to the Orange Live Objects platform. In this case, the controller computes the shortest path between Host1 and the Internet Gateway (Host3): this is illustrated with green in Fig. 10 and goes through OpenvSwitch4 and OpenvSwitch2. The path computation is done by creating an intent that describes a request to the ONOS core to change the functionality of the network. The first intent will choose the shortest past with the lowest delay.

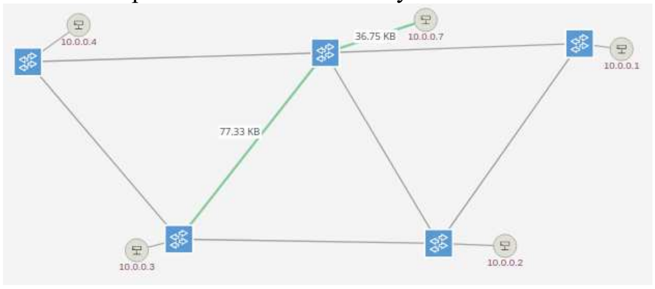
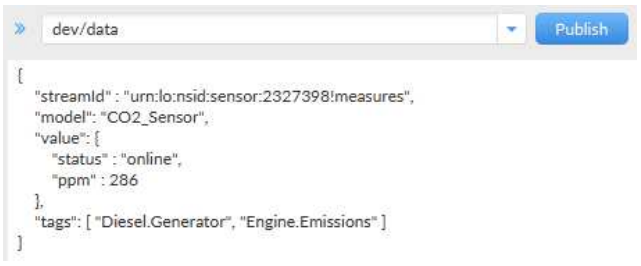


Fig. 10. Shortest path proposed by ONOS

The MQTT.fx client was used to simulate a CO2 sensor that collects engine emissions from a diesel generator. Fig. 11 illustrates the JSON file format in which a measurement is published to the IoT platform.



```
{
  "streamId" : "urn:lo:nsid:sensor:2327398!measures",
  "model": "CO2_Sensor",
  "value": {
    "status" : "online",
    "ppm" : 286
  },
  "tags": [ "Diesel.Generator", "Engine.Emissions" ]
}
```

Fig. 11. CO2 sensor data in JSON format

The data was generated dynamically with different values of the "ppm", so it illustrates fluctuating CO2 levels in the Live Objects dashboard (Fig. 12). Our experiment was performed for one hour (4:25 PM – 5:25 PM).

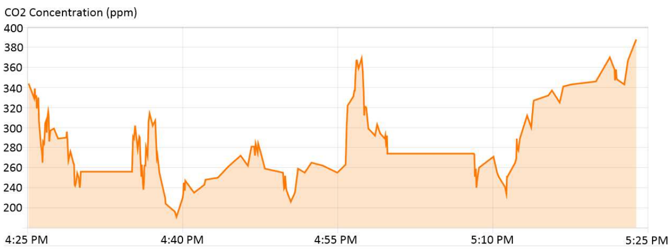
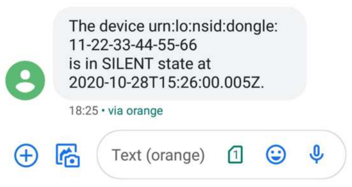


Fig. 12. CO2 levels in Live Objects platform

We also implemented an alarm in Live Objects that sends a SMS after one hour of MQTT.fx inactivity. The experiment ended at 5:25 PM and we received the message alert at 6:25, as illustrated in Fig. 13. The alarm is mandatory in systems that require a continuous connectivity. Next, we wanted to test the ONOS controller capability to detect a system failure and instantly redirect the traffic to an alternative path until the primary link can be used again.


The device urn:lo:nsid:dongle:
11-22-33-44-55-66
is in SILENT state at
2020-10-28T15:26:00.005Z.

18:25 • via orange

Text (orange)

Fig. 13. SMS Alarm in Live Objects

This is an important feature for ensuring fault tolerance in a distributed SDN network. To achieve this, we eliminated the link between OpenvSwitch4 and OpenvSwitch2, used by the shortest path described in the previous experiment. After the link failed, the controller examined the topology to find an alternative path, through OpenvSwitch4, OpenvSwitch5 and OpenvSwitch1, illustrated in green below:

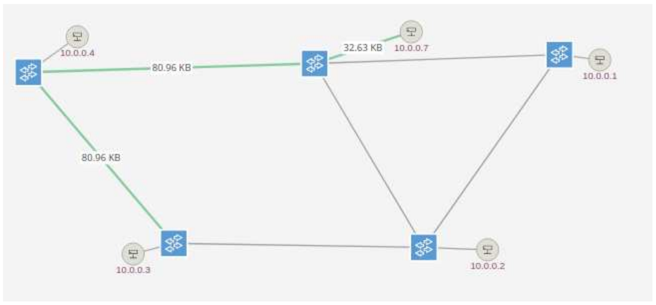


Fig. 14. Alternative path proposed by ONOS

The ONOS controller teaches every switch to send Link Layer Discovery Protocol (LLDP) messages with a probe payload every three seconds, by default, on every port. This additional data explains the increased amount of network traffic between OpenvSwitch4, OpenvSwitch5 and OpenvSwitch1 compared to the one between OpenvSwitch4 and Host3. The hosts do not send any type of discovery messages. When the old link is up the traffic is not directed back to the link where the original intents were installed since the old flow rules were erased with the loss of the connection. The controller will not try to find the optimal path upon new link discoveries, unless forced to do so by a new intent. According to [19], this practice protects against "flickering" events where devices may appear to be switching modes constantly which can cause serious network congestion and resource consumption.

B. *Scenario 2: Mosquitto Client and Broker*

As previously mentioned, the second scenario aims to highlight the option for defining custom forwarding rules in ONOS. In this case the IoT data, generated by the Mosquitto client, needs to be sent to the Mosquitto broker, for edge processing, on the path with the lowest overall latency and processed on the edge. The client simulates a device comprised of a temperature sensor and an actuator for closing a valve in a pharmaceutical plant. The temperature data is sent continuously to the broker where it is monitored. Once the temperature reaches a certain threshold the broker sends a command to the actuator to close the valve. Naturally, this entire process needs to be as quick as possible. In this scenario we used the tool described in [20] to measure the OWDs for each link and compute the path with the lowest delay from Host1 (Mosquitto client) to Host2 (Mosquitto broker). We measured the lowest delay as 0.036 ms through the link defined by OpenvSwitch4, OpenvSwitch3 and OpenvSwitch2. The path is illustrated with green in the Fig. 15:

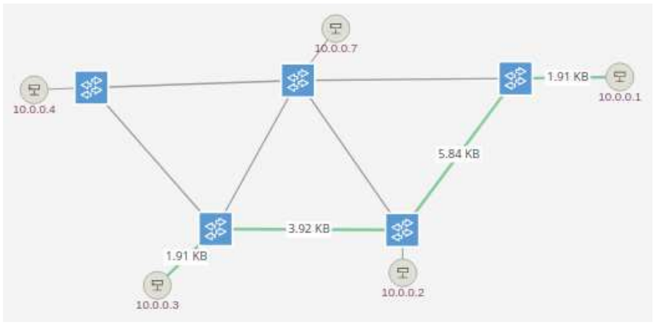


Fig. 15. Lowest delay path between Mosquitto client – Mosquitto broker

The client does not have a graphical user interface and in general offers fewer options than the MQTT.fx. For the purpose of this scenario, several values for the temperature were sent to the broker. In order to publish a message to the "TemperatureTopic", the following CLI command must be used: `mosquitto_pub -h 10.0.0.1 -t TemperatureTopic -m "Temperature is 40 °C, Temperature is 37.5 °C, Temperature is 36.6 °C"`. On the broker side we received the following: `mosquitto_sub -h 10.0.0.1 -t TemperatureTopic Temperature is 40 °C, Temperature is 37.5 °C, Temperature is 36.6 °C`. For the next experiment, we generated cross-traffic from Host4 to Host5 using the iperf tool in order to increase the latency on the link between OpenvSwitch4 and OpenvSwitch3. Consequently, the path with the lowest delay between the Mosquitto client and broker is no longer the one computed in scenario 2. This time we measured the path with the lowest delay as going through OpenvSwitch4, OpenvSwitch1 and OpenvSwitch2 in 0.037 ms. Therefore, we needed to write custom flow rules (see Fig. 16) on each of these switches so that the IoT data is forwarded on this desired path.

```
{
       "priority": 50000,
       "timeout": 40,
       "isPermanent": true,
       "deviceId": "of:0000000000000004",
       "treatment": {
          "instructions": [
            {
              "type": "OUTPUT",
              "port": "1"
            }
          ]
       },
       "selector": {
         "criteria": [
           {
           "type": "ETH_TYPE",
           "ethType": "0x0800"
           },
           {
           "type": IPV4_DST",
           "ip": "10.0.0.1/8"
         },
         {
           "type": IPV4_SRC",
           "ip": "10.0.0.3/8"
         }
       ]
     }
   }
```

Fig. 16. JSON structure of the flow rule

Also, we illustrated the traffic flow changing by deploying a new intent presented in Fig. 17.

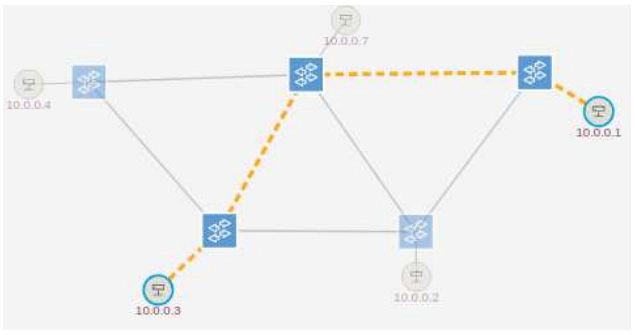


Fig. 17. Creating an intent in ONOS

The JSON rule was tailored for each of the three switches. The new path is illustrated with green in Fig. 18.

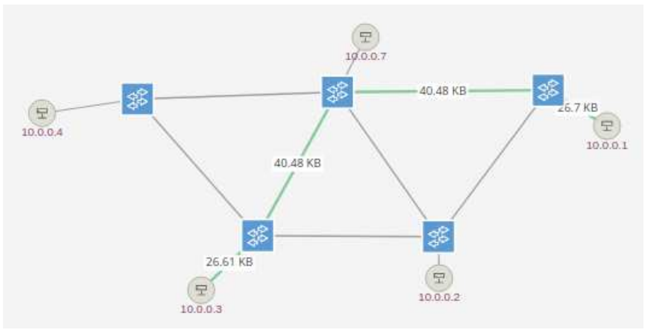


Fig. 18. Mosquitto client – Mosquitto broker new path

V. Conclusions and Future Work

The aim of this paper was to prove that the ONOS controller can be used to build real applications for forwarding traffic in IoT networks. The experiments carried out in Mininet demonstrated that ONOS is easy to deploy and for simple forwarding scenarios, including recovery after a link failure, it is enough to use the predefined forwarding functions. For more complex scenarios such as the ones involving time-critical applications, custom forwarding rules can be created in an intuitive manner using the graphical interface. Future work involves using real sensors and actuators instead of the MQTT clients and adding more cloud-based platforms for data gathering and processing. Also, we will develop more complex forwarding algorithms, which will take into account parameters such as energy consumption, delay and transfer rate in order to compute the best path to these platforms.

References

[1] G. Davis, "2020: Life with 50 billion connected devices," *2018 IEEE International Conference on Consumer Electronics (ICCE)*, Las Vegas, NV, 2018, pp. 1-1, doi: 10.1109/ICCE.2018.8326056.

[2] B. Han, V. Gopalakrishnan, L. Ji and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," in IEEE Communications Magazine, vol. 53, no. 2, pp. 90-97, Feb. 2015, doi: 10.1109/MCOM.2015.7045396

[3] M. Stoyanova, Y. Nikoloudakis, S. Panagiotakis, E. Pallis and E.K. Markakis, "A Survey on the Internet of Things (IoT) Forensics: Challenges, Approaches, and Open Issues," in *IEEE Communications Surveys & Tutorials*, vol. 22, no. 2, pp. 1191-1221, Secondquarter 2020, doi: 10.1109/COMST.2019.2962586.

[4] Y. Chen and T. Kunz, "Performance evaluation of IoT protocols under a constrained wireless access network," *2016 International Conference on Selected Topics in Mobile & Wireless Networking (MoWNeT)*, Cairo, 2016, pp. 1-7, doi: 10.1109/MoWNet.2016.7496622.

[5] O. Bello and S. Zeadally, "Intelligent Device-to-Device Communication in the Internet of Things," in IEEE Systems Journal, vol. 10, no. 3, pp. 1172-1182, Sept. 2016, doi: 10.1109/JSYST.2014.2298837.

[6] A Dhumane, R Prasad, J Prasad, "Routing Issues in Internet of Things: A Survey", Proceedings of the International MultiConference of Engineers and Computer Scientists 2016 Vol I, IMECS 2016, March 16 - 18, 2016, Hong Kong.

[7] I. Bedhief, M. Kassar and T. Aguili, "From Evaluating to Enabling SDN for the Internet of Things," 2018 IEEE/ACS 15th International Conference on Computer Systems and Applications (AICCSA), Aqaba, 2018, pp. 1-8, doi: 10.1109/AICCSA.2018.8612841.

[8] A.H. Eljack, A.H.M. Hassan and H.H. Elamin, "Performance Analysis of ONOS and Floodlight SDN Controllers based on TCP and UDP Traffic," 2019 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE), Khartoum, Sudan, 2019, pp. 1-6, doi: 10.1109/ICCCEEE46830.2019.9071189.

[9] S. Do, L. Le, B.P. Lin and L. Tung, "SDN/NFV-Based Network Infrastructure for Enhancing IoT Gateways," 2019 International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), Atlanta, GA, USA, 2019, pp. 1135-1142, doi: 10.1109/iThings/GreenCom/CPSCom/SmartData.2019.00192.

[10] O. Salman, I.H. Elhajj, A. Kayssi and A. Chehab, "SDN controllers: A comparative study," 2016 18th Mediterranean Electrotechnical Conference (MELECON), Lemesos, 2016, pp. 1-6, doi: 10.1109/MELCON.2016.7495430.

[11] C. Metter, V. Burger, Z. Hu, K. Pei and F. Wamser, "Towards an Active Probing Extension for the ONOS SDN Controller," 2018 28th International Telecommunication Networks and Applications Conference (ITNAC), Sydney, NSW, 2018, pp. 1-8, doi: 10.1109/ATNAC.2018.8615342.

[12] G. Pujolle, "Software Networks: Virtualization, SDN, 5G, and Security, 2nd Edition Revised and Updated", Wiley-ISTE, Feb.2020.

[13] J. Romero Gázquez, and M. Victoria Bueno-Delgado, "Software Architecture Solution Based on SDN for an Industrial IoT Scenario", Wireless Communications and Mobile Computing, vol. 2018, Article ID 2946575, 12 pages, 2018. https://doi.org/10.1155/2018/2946575.

[14] "SDN based Networks for Internet of things: How ONOS helps", Hughes Systique, 2018 [Online], Available: https://hsc.com/Blog/SDN-based-Networks-for-Internet-of-things-How-ONOS-helps.

[15] R. Barrett, A. Facey, W. Nxumalo, J. Rogers, P. Vatcher and M. St-Hilaire, "Dynamic Traffic Diversion in SDN: testbed vs Mininet," *2017 International Conference on Computing, Networking and Communications (ICNC)*, Santa Clara, CA, 2017, pp. 167-171, doi: 10.1109/ICCNC.2017.7876121.

[16] ONOS, 2020, [Online], Available: https://wiki.onosproject.org/display/ONOS11/Experimental+Features.

[17] Live Objects, 2020, [Online], Available: https://liveobjects.orange-business.com/doc/html/lo_manual.html#OVERVIEW.

[18] Mosquitto, 2020, [Online], Available: https://mosquitto.org/.

[19] Mirantis, 2020, [Online], Available: http://trainer.edu.mirantis.com/SDN50/sdn.html.

[20] A. Taut, I.A. Ivanciu, E. Luchian, and V. Dobrota, "Active Measurement of the Latency in Cloud-Based Networks", ACTA TECHNICA NAPOCENSIS, Electronics and Telecommunications, ISSN 1221-6542, Vol.58, No.1, 2017, pp.22-30.