Fast-Failover Groups for SDN Based Horiontal Internet of Things Architecture

MuthuKumaraSwamy Sathananth, Viswanath
viz@dal.ca

Khandekar, Shrinivas
shrini@dal.ca

ABSTRACT

The internet architecture was not developed with the Internet of Things (IoT) in mind. The advent of IoT has increased the need for a communication platform that allows multiple devices, access technologies (WiFi, Bluetooth and so on), network infrastructures to coexist. This model is referred to as the horizontal model of IOT. Software Defined Networking (SDN) and its network programmability have been viewed as a potential environment to form the platform for this type of communication. Recently, a few researchers have proposed architectures for Horizontal IoT using the Network Operating System (NOS) offered by SDN as the core. This paper presents an implementation of a Fast Fail-over mechanism in a Horizontal IoT environment to enhance the reliability of the communication medium. We are also providing an implementation of this to prove the feasibility.

Keywords

Internet of Things(IOT), Horizontal Model,Software Defined Networking(SDN),Raspberry Pi

1. INTRODUCTION

The evolution of communication technologies allowed handheld devices to stay connected to the Internet and communicate with each other by using the Internet as the medium. This communication gave rise to the Internet of Things (IoT). The Internet is no longer constituted by just routers, switches, and computers. Devices as small as a watch to ones as big as an airplane are also a part of the Internet. The result of this growth was heterogeneity in the characteristics of devices that are connected to the Internet.

1.1 Vertical IoT Model

The heterogeneity also brought a demand for a common communication medium for all devices to interact. The requirement was fulfilled by the emergence of Cloud Computing. But, the problem with this solution is that, the Cloud Service Provider (CSP) supported only a few devices. Very often the devices that were supported by a cloud storage were owned by the company that owns the cloud. This model of the IoT is referred to as the vertical model.[7]

One of the main advantages of the vertical model is that the end user need not worry about the compatibility of the devices used. Since all the devices in the environment are owned by a single company, The company makes sure that these devices are compatible with each other. The protocols are developed by the owner.

The main disadvantage of a vertical IoT model is the lack of interoperability. Devices from one company were not able to interact with the devices offered by another company. The protocols used by every company is different. Also, the difference in data formats used by these devices made it impossible for the interaction to happen. The sensors and actuators support by each device varied. These difference made innovation and development platform dependent.

1.2 Horizontal IoT Model

To solve these problems, a horizontal model for IoT was proposed. The horizontal model allowed devices that operate on different platforms and architectures to interact with each other through a common framework. The aim of this model was to allow the developers to integrate components from different vendors to provide a solution. One of the main advantages of this model is its interoperability. The horizontal models allow the reuse of programs in the form of interfaces. The data and the methods for processing the data are also open to everyone.

The implementation of the horizontal model required a common platform for the devices to interact. One of the earliest solutions was a Machine-To-Machine architecture (M2M).[8] M2M architecture allowed multiple devices to communicate with each other by means of a standardized platform. The problem of interoperability was solved to an extent, but the interoperability of different data models could not be supported. Other solutions like an IoT community were also proposed. Services which are similar were organized in a community and each community followed a standard. But this was viewed as a simple extension of the vertical model. The model was neither open to the users nor convenient for introducing new applications and services.

Software Defined Networking (SDN) separates the control-plane (decisions about how to handle the traffic) from the data-plane (devices that handle the traffic). SDN uses a single software called the SDN Controller to control different data-plane elements. This feature of SDN encouraged the usage of SDN in implementing the horizontal model of IoT. NOS can be used as a common platform that connects to different devices through the SDN South Bound Interface (SBI). Different IOT applications can interact with the SDN controller through the SDN North Bound Interface (NBI). The controller controls and coordinates the communication of the applications with the devices thereby making it a horizontal model.

IoT has many applications, a few of these applications might need an assurance that the information sent by one device will be received by another. When the underlying network size is reasonable, there is the probability of having failed nodes in the path from the source device to the destination device. These failures can lead to interruption in communication. Sometimes the cost paid for this interruption can be significantly high.

For example, let us consider the case of a home automation system. The systems keeps track of who is entering a room. It restricts people from entering some rooms. Gives access to only a few people in the house. Let us assume a scenario where the room which has cash has access to only the owner. The owner needs some case to a person and there is a emergency. He is trying to unlock a room, but due to node failures, he is not able to open the room. Usability rules state that if an application should be successful it should make the user wait for a long time. Let us consider the case of playing a LAN game. While playing the game, due to some failure in the intermediate nodes there is a notable delay in communication. This can result in a person losing the game. These delays irritates the person using these devices. To make the information reaches the destination in time, we should be able to fix these failures in a short span of time. To ensure these failures are detected and fixed in a short span of time, we need Fast-Failover Groups (FFGs).

To make sure an information from the source reaches the destination, Fast-Failover groups are generally used in SDN. Fast-Failover groups enable a switch to store multiple paths to the destination. The first entry of the group represents the primary path and all the other entries represent the alternate paths. When the primary path fails the information is routed through one of the alternate paths. The objective of our project would be to implement a Fast-Failover group in an IoT environment(Raspberry Pi).

2. RELATED WORK

A research paper published by Mineraud et al.[9] identified some of the gaps in IoT platforms. One of the main gaps was integration between sensing and actuating devices. In the absence of a communication standard, each device supported a different subset of communication protocols and interaction patterns. This paper also stressed the need for a standardization of the communication protocols used and the need for interoperability between heterogeneous devices. This leads to the development of the horizontal model of IoT.

One of first few models proposed was the Machine to Machine (M2M) model proposed by Floeck et al. [8] A horizontal model was implemented by gathering requirements from different vertical models and creating a common reference architecture and protocols. Some examples of this model are ETSI M2M and One M2M which is a successor of ETSI M2M. There were a few challenges in an M2M design. Firstly, since it involved many sources, the aggregation of information from diverse sources was difficult. Secondly, the horizontal model of IoT required complete interaction between all stakeholders. This interoperability among different stakeholders was very difficult to achieve.

Asaeda et al. [12] proposed a community-based approach. They made use of a community-based Information Centric Network to implement the horizontal model of IoT. Data Clouds were used to represent different communities. Each community consisted of users with common interests in data and information. The Internet was viewed as a collection of cloudlets each representing a community. But, the suggested architecture was not open to the user. This hampered innovation and made it difficult to develop applications.

In the recent past, several attempts have been made to use SDN as the common framework for communication between heterogeneous devices. SDN makes use of a centralized routing model, the controller computes all the routes. This centralized calculation makes SDN unsuitable for low power devices. 6TiSCH combines SDN-type centralized routing, for time-sensitive flows with Routing Protocol for Low-Power and Lossy Networks (RPL).[11] RPL is a very important protocol for low power wireless networks. The network is considered as a tree. Every node keeps track of an immediate one-hop parent. The parent is chosen based on an Objective Function (OF). RPL allows the nodes to quickly send information up the tree.

Baddeley et al.[6] proposed micro SDN which is a lightweight implementation of the actual SDN. It optimized the existing SDN environment to make it suitable for use in the wireless environment. This approach involved architectural, memory and controller optimizations. Some of the architectural optimizations are the use of source routing to prevent unwanted control messages, timers to refresh flow tables to reduce the number of control messages. Memory optimizations involve reuse of flow table entries and lower buffer sizes. Along with all these optimizations, an embedded controller was also added to handle simple requests. The intention of this addition was to reduce the number of messages to the controller.

Tanha et al.[10] made a comparison between reactive and proactive protocols in the wireless environment. Reactive protocols install new routes when a failure is detected. Proactive protocols like Fast Fail-over groups install alternate routes before failure to protect the network. Proactive protocols exchange a lot of control messages to keep the network state updated. This causes higher interference in the network. Therefore, the throughput of the network in the wireless environment is lesser when compared to reactive protocols.

3. IMPLEMENTATION

The implementation of Horizontal SDN was carried out using a real test bed. The testbed was made of two switches and two hosts. Both switch and host are configured on a Raspberry Pi (RPi). The difference between a host and switch is, switch ran OpenVSwitch and hosts did not. The following section discusses some of the features of RPi and different versions of RPi.

3.1 Raspberry Pi

Raspberry Pi is a series of small single-board computers. There are several models of Raspberry PI available, and they are split into three categories namely Model A, Model B, Model Zero. Following are some of the specifications of these models.

3.1.1 Model A

- Gereneration - 1 and 1+
- GPU - Broadcom VideoCore IV @ 250 MHz, OpenGL ES 2.0, MPEG-2 and VC-1 1080p30 H.264/MPEG-4 AVC high-profile decoder and encoder
- Instruction Set - ARMv6Z (32-bit) USB 2.0 ports - 1
- Video input - 15-pin MIPI camera interface
- Video Output - HDMI
- Power - 5 V via MicroUSB or GPIO header

3.1.2 Model B

- Gereneration - 1 and 1+, 2, 2 ver 1.2, 3, 3+
- Instruction Set - ARMv6Z (32-bit), ARMv7-A (32-bit), ARMv8-A (64/32-bit)
- GPU - Broadcom VideoCore IV @ 250 MHz, OpenGL ES 2.0, MPEG-2 and VC-1 1080p30 H.264/MPEG-4 AVC high-profile decoder and encoder
- USB 2.0 ports - 2 or 4
- Video input - 15-pin MIPI camera interface Video Output x - HDMI
- Power - 5 V via MicroUSB or GPIO header

3.1.3 Model Zero

- Generation - PCB ver 1.2, PCB ver 1.3 and Wireless W
- Instruction Set - ARMv6Z (32-bit)
- GPU - Broadcom VideoCore IV @ 250 MHz, OpenGL ES 2.0, MPEG-2 and VC-1 1080p30 H.264/MPEG-4 AVC high-profile decoder and encoder
- USB 2.0 ports - 1 Micro-USB port
- Video input - MIPI camera interface (not available in PCB ver 1.2)
- Video Output - Mini-HDMI
- Power - 5 V via MicroUSB or GPIO header

These three series of RPi has a range of models. A comparison between the top models of each model can be seen in Table 1.

We will be using Raspberry PI 3 Model B+ for our implementation, Raspberry PI 3 has wide range of features a few of them are

- Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC @ 1.4GHz
- 1GB LPDDR2 SDRAM
- 2.4GHz and 5GHz IEEE 802.11.b/g/n/ac wireless LAN, Bluetooth 4.2, BLE

Figure 1: An image of Raspberry Pi 3 Model B+ [5]

- Gigabit Ethernet over USB 2.0 (maximum throughput 300 Mbps)
- Extended 40-pin GPIO header
- Full-size HDMI
- 4 USB 2.0 ports
- CSI camera port for connecting a Raspberry Pi camera
- DSI display port for connecting a Raspberry Pi touch-screen display
- 4-pole stereo output and composite video port
- Micro SD port for loading your operating system and storing data
- 5V/2.5A DC power input
- Power-over-Ethernet (PoE) support (requires separate PoE HAT)

3.2 Raspbian

Raspbian is an free Operating System based on Debian, a highly recommended OS for Raspberry PI which optimises RPi hardware. Raspbian comes with variety of precompiled software bundles which can easy installed on Raspberry Pi. Raspbian has a community, which emphasis on improving the stability and performance.

3.3 Open vSwitch

Open vSwitch is a multilayer software switch, which implements a production quality switch platform that supports standard management interfaces and opens the forwarding functions to programmatic extension and control.[4]

We will be using Open vSwitch in RPi to function as a virtual switch in VM environments. In addition to exposing standard control and visibility interfaces to the virtual networking layer, it was designed to support distribution across multiple physical servers.

3.4 Proposed Architecture

The proposed architecture for the RPI based SDN testbed was as shown in the figure 2. In the architecture of 6 RPis, 3 were configured as hosts and the other 3 were configured as switches. Each host was connected to a dedicated switch through which it will send and receive the packets. Switches had connections between each other and a connection with

Specification	Raspberry PI 1+	Raspberry PI 3+	Raspberry PI WH
Instruction set	ARMv6Z (32-bit)	ARMv8-A (64/32-bit)	ARMv6Z (32-bit)
SoC	Broadcom BCM2835	Broadcom BCM2837B0	Broadcom BCM2835
CPU	1 ARM 700 MHz	4 Cortex-A53 1.4 GHz	ARM 1 GHz single-core
Memory (SDRAM)	512 MB	1 GB	512 MB
USB 2.0 ports	1	4	1 Micro-USB
On-board storage	MicroSDHC slot	MicroSDHC slot and USB Boot Model	MicroSDHC slot
Video input	15-pin MIPI	15-pin MIPI	MIPI
Video outputs	HDMI	HDMI	Mini HDMI
Power ratings	200 mA	459 mA	100 mA
Power source	5 V	5 V	5 V
Size	65 mm x 56.5 mm x 10 mm	85.60 mm x 56.5 mm x 17 mm	65 mm x 30 mm x 5 mm
Price	USD 20	USD 35	USD 10
Type	Model A	Model B	Zero

Table 1: RPi Comparison

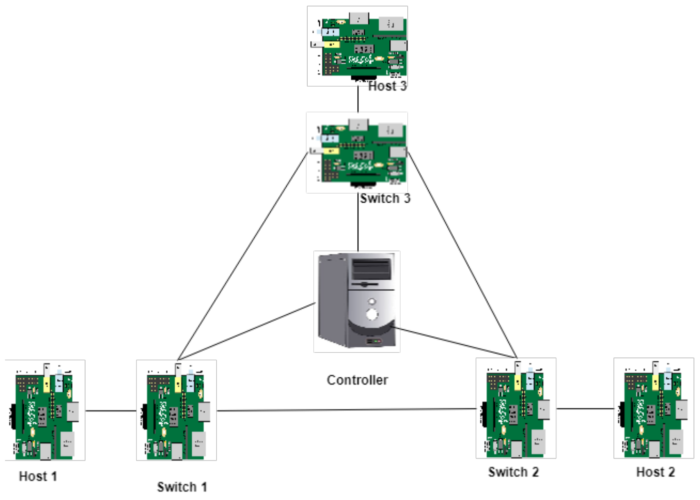


Figure 2: Proposed architecture of the testbed [3] [2]

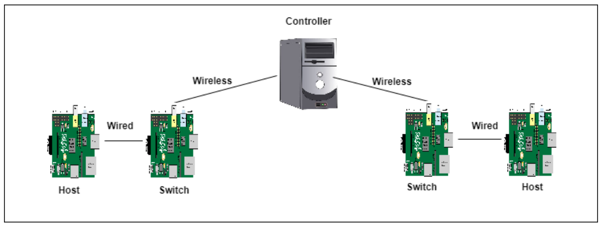


Figure 3: Implemented Architecture [3] [2]

the controller. The switches had wired/wireless connections with the controller.

Each switch consists of one or more flow tables and a group table, which perform packet lookups and forwarding, and provide a secure communication channel between the controller and other switches in the network. We will be configuring a workstation as a controller, which will manage the network topology, flow control, statistics, device and network management.

In the architecture shown in Figure 2, Host 1 is connected to Switch 1, Host 2 is connected to Switch 2 and Host 3 is connected to Switch 3. All switches are connected with each other and the controller.

All the switches used in this architecture were supposed to make use of Open VSwitch (OVS) version 2.5.5. The controller we planned to use is a RYU controller.

3.5 Implemented Architecture

We started with the architecture shown in Figure 3. We used four RPis, out of which two were configured as hosts and the other two were configured as switches. Each host is connected to a dedicated switch through wired interfaces. Switches are connected to the controller through a wireless interface. A windows 10 machine was used as the controller.

Initially, the version used was OVS v2.5.5. The ports user were integer values. Due to some problems explained in the following section we had to move to version 2.9.2 which is more modern. RYU version 4.26 was used as the controller.

4. TEST EVALUATION

The parameters considered to evaluate the testbed shown above will be throughput and delay. Similar evaluations will be done for the same network in Mininet, RPi wireless communication, and wired communication environments. The results obtained from these experiments will be compared to obtain an inference.

5. CHALLENGES AND FUTURE WORK

The testbed showed in Figure 3 was implemented. With the RYU controller running on Window 10 machine. While testing the architecture lots of obstacles were faced. We have classified the obstacles faced during the implementation of the project are classified into four categories.

5.1 Hardware Limitations

The hardware used in this architecture is of a low-cost and low-power system. Since RPi is a single board computer, there were many resource limitations. Some of them are as follows:

5.1.1 Number of LAN ports

RPi has a single LAN port, which restricts us to have only one wired connection to another machine, without using any USB to LAN adapters. In the proposed architecture shown in Figure 2, it can be noticed that there are 3 connections originating from each RPi. To implement these connections at least 2 USB to LAN adapters are required.

5.1.2 Wi-Fi Adapters

If the architecture shown in Figure 2 is implemented in a wireless environment, then every RPi needs to have three simultaneous network connections, demanding more wireless interfaces. This could be achieved through USB to WiFi adapters. Each board required a minimum of 2 such adapters.

5.1.3 RPi Version

The installation of OVS version 2.5.5 had few features that relied on the underlying hardware. When RPi B 2 was used as a switch, there were problems with OVS installation process. The issues were resolved by switching to the RPi B 3+.

5.2 Protocol Difference

With the limitations of resources, a combination of both wired and wireless communication was used. Switches were connected to the controller using wireless medium and the connection between two switches was wired, preference was given to wired connection over the wireless connection. Therefore the controller did not receive any packets. Instead, all the packets were delivered to the destination through the wired connection.

5.3 Topology of Network

The RYU controller is supposed to return a set of links depending upon the topology of the network. During the implementation, every time the controller returned empty links. This can be attributed to some problem in the RYU

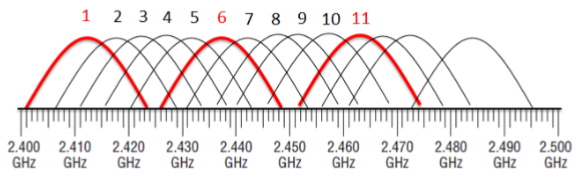


Figure 4: 2.4 GHz Band [1]

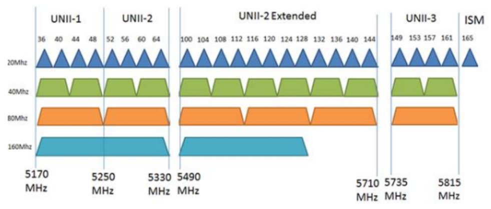


Figure 5: 2.4 GHz Band [1]

environment. Since the links were empty, the destination ports could not be identified.

5.4 Wireless Environment

There were a few limitation that were caused due to the wireless environment, namely

5.4.1 Firewall issues

The Windows 10 Firewall blocked the connection made through the router. Both ICMP V4 and ICMP V6 packets were not allowed to enter through the firewall. Therefore no packets were received at the controller.

5.4.2 Bandwidth

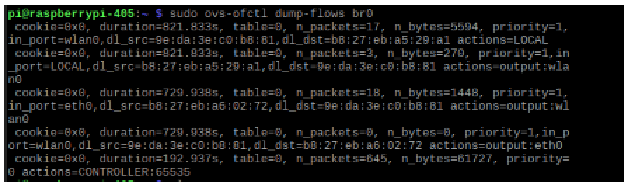
Modern Wi-fi devices support two bandwidths, namely 2.4GHz and 5GHz bandwidth. As shown in Figure 4, the maximum number of total bands that can co-exist is 3 in 2.4GHz. When the number of bands exceeds 3 there is interference in the signals. Whereas the 5GHz band supports up to 21 simultaneous WiFi networks. This can be seen in Figure 5.

5.4.3 Access Points

In the wireless environment, each switch should connect to two other switches through the wireless medium. This constrained the RPi to act both as a switch and access point. This had a significant impact on the performance of the RPi. OVS alone consumes most of the Central Processing Unit (CPU) usage, and operating as an access point will make it bulkier.

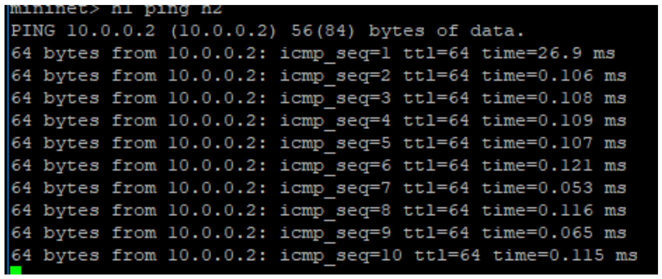
It can be clearly observed from the Figure 6, that the switch was successfully able to capture the flow entries to and from the controller and entries to and from the host, where the port LOCAL represents the host.

Due to the above reasons, the project is still in progress. Although there were flow entries, the packets were not forwarded properly. Only one-way communication is established in the stipulated time. In the future, we plan to research the architecture and implement FFG to assure fast recovery.



```
pi@raspberrypi-405:~ $ sudo ovs-ofctl dump-flows br0
 cookie=0x0, duration=821.833s, table=0, n_packets=17, n_bytes=5594, priority=1,
in_port=wlan0,dl_src=9e:da:3e:c0:b8:81,dl_dst=b8:27:eb:a5:29:a1 actions=LOCAL
 cookie=0x0, duration=821.833s, table=0, n_packets=3, n_bytes=270, priority=1,in
_port=LOCAL,dl_src=b8:27:eb:a5:29:a1,dl_dst=9e:da:3e:c0:b8:81 actions=output:wla
n0
 cookie=0x0, duration=729.938s, table=0, n_packets=18, n_bytes=1448, priority=1,
in_port=eth0,dl_src=b8:27:eb:a6:02:72,dl_dst=9e:da:3e:c0:b8:81 actions=output:wl
an0
 cookie=0x0, duration=729.938s, table=0, n_packets=0, n_bytes=0, priority=1,in_p
ort=wlan0,dl_src=9e:da:3e:c0:b8:81,dl_dst=b8:27:eb:a6:02:72 actions=output:eth0
 cookie=0x0, duration=192.937s, table=0, n_packets=645, n_bytes=61727, priority=
0 actions=CONTROLLER:65535
```

Figure 6: 2.4 GHz Band [1]



```
mininet> h1 ping h2
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=26.9 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.106 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.108 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.109 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.107 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.121 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.053 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.116 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.065 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=0.115 ms
```

Figure 7: 2.4 GHz Band [1]

The same code when executed in Mininet gives perfect results. So it is evident that the problem is in the underlying hardware. The results of the ping in mininet are shown in figure 7.

6. CONCLUSION

We presented RPi Testbed, a low-power and low-cost center for the SDN-enabled network. We also demonstrated how economical computers such as RPI's can be used to mimic a network of OpenFlow switches in the SDN-enabled environment. The difficulties in identifying the hardware and the hybrid connection made the architecture difficult to implement, although the testbed was set up. Flow entries were successfully inserted. In the future, the reason for the absence of links should be identified and fixed, allowing us to implement complex networks.

7. REFERENCES

[1] Faq - what is the difference between 2.4 ghz and 5ghz? Accessed July 14, 2018.

[2] Flaticon, the largest database of free vector icons. Accessed July 14, 2018.

[3] Free flowchart maker and diagrams online. Accessed July 14, 2018.

[4] Production quality, multilayer open virtual switch. Accessed July 14, 2018.

[5] Teach, learn, and make with raspberry pi. Accessed July 14, 2018.

[6] M. Baddeley. Evolving sdn for low-power iot networks. 6 2018. IEEE International Conference on Network Softwarization, NetSoft 2018 ; Conference date: 25-06-2018 Through 29-06-2018.

[7] Y. Li, X. Su, J. Riekki, T. Kanter, and R. Rahmani. A sdn-based architecture for horizontal internet of things services. In *2016 IEEE International Conference on Communications (ICC)*, pages 1–7, May 2016.

[8] M.Floeck, A.Papageorgious, and A. Schuelke. Horizontal m2m platforms boost vertical industry: effectiveness study for building energy management systems. *2014 IEEE World Forum on Internet of Things*, pages 15–0, 2014.

[9] J. Mineraud, O. Mazhelis, X. Su, and S. Tarkoma. A gap analysis of internet-of-things platforms. *CoRR*, abs/1502.01181, 2015.

[10] M. Tanha, D. Sajjadi, and J. Pan. Demystifying failure recovery for software-defined wireless mesh networks. 01 2018.

[11] P. Thubert, M. R. Palattella, and T. Engel. 6tisch centralized scheduling: When sdn meet iot. In *2015 IEEE Conference on Standards for Communications and Networking (CSCN)*, pages 42–47, Oct 2015.

[12] T.Luo, H.P.Tan, P. C. Quan, Y. W. Law, and J. Jin. Enhancing responsiveness and scalability for openflow networks via control-message quenching. *International Conference on ICT Convergence*, pages 348–353, 2012.