Dynamic QoS support for IoT backhaul networks through SDN

Stefanos Peros, Hassaan Janjua, Sven Akkermans, Wouter Joosen, Danny Hughes
imec-DistriNet
KU Leuven
Leuven
B-3001, Belgium
Email: firstname.lastname@cs.kuleuven.be

***Abstract*—The Internet-of-Things (IoT) is growing significantly, becoming an integral part of daily life. The services provided by the IoT are often dynamic and driven by physical events. When certain events occur, Quality of Service (QoS) requirements for services can change. For instance, a video camera may need to transmit higher quality footage over a network when motion is detected. A flexible network is therefore required that reacts to events and re-configures itself dynamically to meet dynamic QoS constraints. In contrast to traditional networks, Software-Defined Networking (SDN) decouples the data plane and the control plane, resulting in a programmable network. Programmable networks enable the development of SDN applications that can be used to meet the dynamic QoS needs of present-day IoT applications. In pursuit of this vision, this paper presents the architecture and implementation of a framework, built with SDN, to support dynamic QoS for the IoT. We demonstrate the framework's ability to adapt to dynamic QoS requirements and its performance through experiments on a hybrid network topology.**

I. Introduction

The Internet of Things (IoT) is growing rapidly, both in industry and academia [1]. It consists of everyday devices that interact with each other, using wireless networks, and cooperate with their neighbors to reach common goals.

Applications are built on top of the IoT to enhance the quality of life in different domains [1]. These applications require Quality-of-Service (QoS) guarantees from the network to perform their tasks when events occur. An intrusion detection system, for example, typically consists of several video cameras and motion sensors distributed over an area. When a motion sensor detects irregular movement, the camera in the same location starts transmitting footage to the interested stakeholders. If the network cannot assign enough resources to the video camera when the intruder is detected, the quality of the footage is compromised. In this paper, we consider three QoS parameters that are influenced by the network: transmission delay, bandwidth and packet loss.

In traditional IP networks, QoS support is static. Fixed network resources are assigned to applications upon deployment. This is insufficient for dynamic IoT applications, since their QoS requirements cannot be specified in advance [2]. Therefore, dynamic QoS support requires mechanisms to re-configure the network to allocate network resources to devices on demand [3]. A global network topology overview and network programmability are key elements to implement dynamic QoS support. This is an excellent fit with the characteristics of Software-Defined Networking (SDN).

In SDN, control logic is centralized and decoupled from the forwarding devices [4]. External network applications can re-configure the network dynamically through interfaces that are provided by the control layer. As a result, core network functionalities, such as a firewall, can be implemented in software.

Research has been conducted into frameworks that implement QoS support on top of SDN [5], [6], [7], [8], [9], [10]. However, the proposed frameworks are designed for services with static QoS behavior, require technical knowledge and do not support dynamic QoS requirements.

This paper takes a step towards supporting dynamic QoS for IoT applications. A novel SDN framework is proposed that enables application users and developers to specify dynamic QoS rules for their IoT devices. For that purpose, existing research in QoS and typical industrial IoT applications are analyzed to create a classification of default QoS profiles for IoT devices. The proposed framework uses a default QoS profile classification to limit the technical knowledge required by the end-user, enabling treating the IoT as a commodity. To the best of our knowledge, in SDN, this is the first framework that addresses QoS requirements for IoT devices and masks technical details from users by means of default QoS profiles. An experiment evaluates the framework for a hybrid network topology. The results illustrate the ability of the framework to ensure QoS by measuring the bandwidth, latency and packet loss before and after network re-configuration.

The rest of the paper is organized as follows: Section 2 presents a high level overview of the framework architecture and the default QoS profile classification. Section 3 explains the key abstractions, along with the implementation techniques and technologies. Section 4 presents the evaluation of the framework. Section 5 describes related work. Section 6 concludes the paper. Section 7 discusses future work.

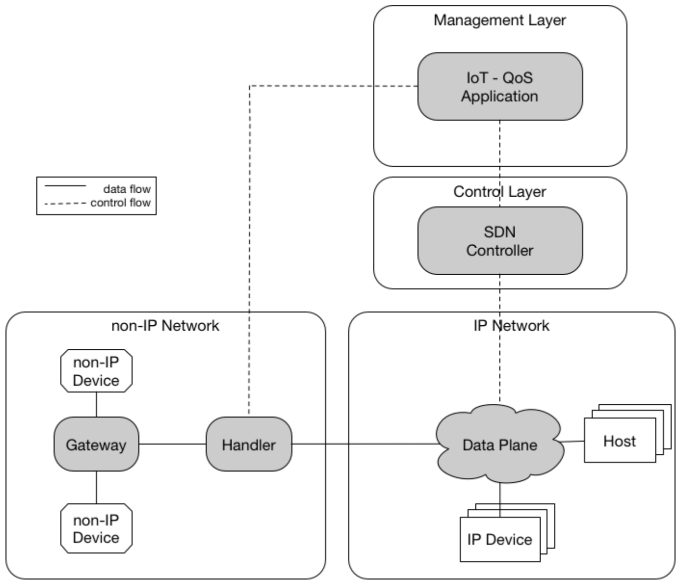

Fig. 1. System architecture overview.

II. Architecture

The architecture of the proposed framework consists of four subsystems: non-IP networks, IP networks, control and management layer. Figure 1 shows the subsystems, along with the communication channels that connect them, namely the data and the control flows.

Based on their communication capabilities, IoT devices can be classified into non-Internet Protocol (non-IP) and Internet Protocol (IP) devices. The non-IP Network consists of the non-IP devices and their gateway. These non-IP devices send data, through the gateway, to client applications that run on hosts in the IP network. The Gateway component forwards the data to the Handler, which can mark the packets before injecting them to the network. This is necessary to make them distinguishable, since they all share the same source IP address. The IP Network consists of three components: the data plane, IP devices and traditional network hosts. Client applications, such as a weather system, typically run on these hosts and receive data from the IP and non-IP devices. The data plane is a collection of OpenFlow-enabled switches [11] that are managed by the SDN controller in the control layer.

The Control Layer is responsible for managing the data plane and providing external network applications with an API to interact with the network. This functionality is implemented by the SDN controller. The Management layer consists of the IoT-QoS Application component, which: (1) detects new devices that join the network, (2) enables users to specify types to their devices, which determine the assignable QoS profiles, (3) calculates QoS paths between pairs of devices, (4) re-configures the network dynamically to ensure QoS and, (5) allows the specification of dynamic QoS rules, enforcing specific QoS levels only when events are triggered. The events that trigger dynamic QoS are linked to a non-IP Device and become activated when that device matches the user-specified criteria.

TABLE I
Default QoS profiles.

	QoS level	Bandwidth (Mbps)	Latency (ms)	Packet Loss (%)
Video	Low	2,8	15	2,5
	Medium	4,8	10	2
	High	7,2	5	1
Sensor	Low	n.a	500	2
	Medium	n.a	300	1
	High	n.a	150	0
Actuator	Low	n.a	1000	2
	Medium	n.a	1000	1
	High	n.a	100	0
Audio	Low	0,064	30	25
	Medium	0,248	10	13
	High	1.4	3	1

A. Default QoS Profiles

A default QoS profile classification is proposed for IoT devices to enable non-technical users to use the framework without requiring technical knowledge. In particular, IoT devices are classified into four types: sensor, actuator, video and audio. Table I shows a complete overview of the proposed classification. Traffic produced by sensors and actuators does not need high bandwidth guarantees. Instead, QoS for these two device types focuses on packet loss and latency. In [12], the authors differentiate between high priority and low priority sensor network traffic and measure the average delivery delay in each case. Audio and video devices [13] require bandwidth guarantees from the network, together with packet loss and latency. For audio devices, research has been done in [14] for the requirements in latency: sounds are perceived as separate when there is a delay greater than 30ms . For packet loss values, we used VoIP as a reference [15]. For video traffic, Ongaro et al. [6] propose a formula that maps latency and packet loss values to Mean Opinion Score (MOS) values, in order to measure the user-perceived QoS. The required bandwidth depends on the camera resolution and stream encoding. The proposed values assume a video resolution of 1.3MP and H.264 encoding.

III. Implementation

This section describes the implementation details of the framework. First, we explain the key abstractions that were used to implement the framework. After, the tools, technologies and techniques that were used to implement the proposed architecture are discussed.

A. Key Abstractions

Figure 2 shows the key abstractions used to model the domain and implement the framework: QoS Rule, QoS configuration, Event and Device. A QoS Rule represents an event-driven rule that causes the system to enforce QoS between two devices in the network. It consists of a source and destination device, the triggering event and a QoS configuration. An Event is linked to a device, often a sensor, and becomes activated when the sensor reading exceeds a threshold. Devices are IoT devices in the network and are further classified into

IP and non-IP, based on their communication capabilities. Finally, a QoS Configuration represents a collection of values for bandwidth, packet loss and latency. Every QoS Rule is linked to a QoS configuration, which specifies QoS guarantees that the framework needs to meet, between the source and destination Device, when the associated Event is activated.

We illustrate the steps that the user follows in order to create a dynamic QoS rule, through a GUI front end, to demonstrate the use of these abstractions. In this example, the goal of the user is to create the following QoS rule: when the non-IP motion sensor detects movement, the IP camera traffic, destined for the video client, should be assigned a high QoS. This QoS rule can be part of a larger use-case, i.e. an intrusion detection application, where camera footage quality changes when an intruder is detected in the protected area. To achieve this goal, the first step for the user is to create the movement detection event, by selecting the motion sensor and specifying a threshold. The next step is to create a QoS rule which links the traffic between the IP camera and video client to that event. The final step is to assign a QoS configuration to this QoS rule, i.e. high QoS for video traffic. More concretely, the front-end dynamically obtains information about the devices that are present in the network, including the IP camera and non-IP motion sensor, and presents them to the user so that no manual registration is necessary. The user navigates to the Events tab to create a motion detection event. For that purpose, the user selects the motion sensor from a drop-down list and specifies a threshold at which an event is triggered. The event is then registered in the back-end. The user then links this event with the QoS rule between the IP camera and the video client. To do this, the user navigates to the QoS Rules tab, where he creates a new QoS Rule. The user selects the IP camera as the source device, the video client as the destination and the motion-detection event as the trigger. Finally, the user selects a QoS configuration from a drop-down list, obtained dynamically from the back-end based on the type of the source device, in this case high QoS configuration for video. These configurations consist of predefined values for bandwidth, packet loss and latency, summarized in Table I. This requires little knowledge from the user about concrete QoS parameter values and eases profile selection for the device. To complete the use-case, the user selects the ”video high QoS” configuration and the back-end registers the QoS rule.

B. Tools and Technologies

The goal of this implementation is to build a system that enables users to create dynamic QoS rules for their IoT devices, without requiring technical knowledge. This entails that the system can detect devices within a network, both IP and non-IP. When events trigger the system, it is responsible for re-configuring the network on-the-fly to reallocate network resources between devices, to meet their dynamic QoS constraints. Users can interact with the system and specify QoS rules through a front-end.

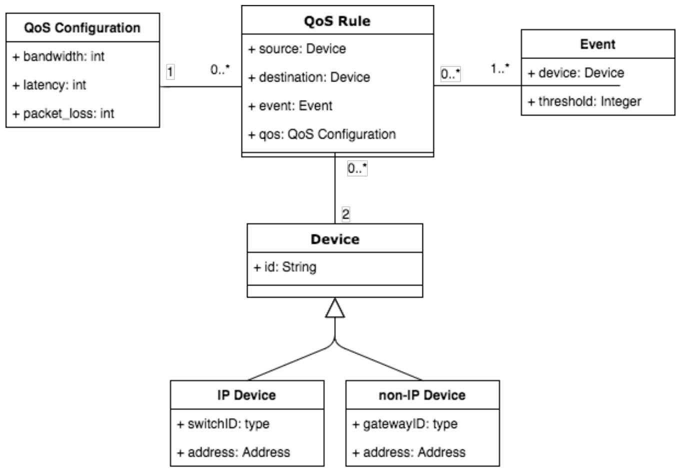

Fig. 2. Domain model.

For the non-IP Network, a SAMD21 Cortex-M0+ 32bit low power ARM MCU chip is used to host the Gateway component. The chip reads sensor values once per second and forward them serially, through a USB cable, to the Handler, implemented on a Raspberry Pi 3. Sensor values are obtained from two off-the-shelf non-IP devices: an analog temperature and a motion sensor. These are connected with the GPIO pins of the MCU. The Handler parses the received sensor readings, processes them and wraps them in IP packets that are transmitted over the IP Network. The underlying challenge is that all data sent by non-IP devices appear to arrive under the Handler's IP address in the wired network. It is thus not possible to differentiate these packets based on source IP address to enforce QoS on a per-device level. To solve this issue, the Handler registers non-IP devices in the sensor network using a REST API interface provided by the IoT-QoS Application. In turn, the IoT-QoS Application assigns a unique identifier to each registered non-IP device, stored in a table. When QoS needs to be enforced for a non-IP device, the Handler modifies the Type of Service (ToS) IP header field for the non-IP device, based on the unique identifier, to distinguish its traffic in the network. Finally, the IoT-QoS Application installs flow-rules to the SDN switches in the Data Plane that match packets on the ToS field.

The IP Network is implemented using Mininet [1], a network emulator tool. The Hosts and IP Devices reside within the network as virtual machines. The Data Plane consists of SDN forwarding devices implemented in software using Open vSwitch (OvS), an open-source tool [2].

The control layer is implemented using Floodlight, an open-source SDN controller [3]. Floodlight has a modular architecture and provides external applications with a REST API [16] to configure the network. Additionally, it supports the hybrid network topology used in this implementation, a combination

[1]Described at http://mininet.org/overview/
[2]Described at http://openvswitch.org/
[3]Described at http://www.projectfloodlight.org/floodlight/

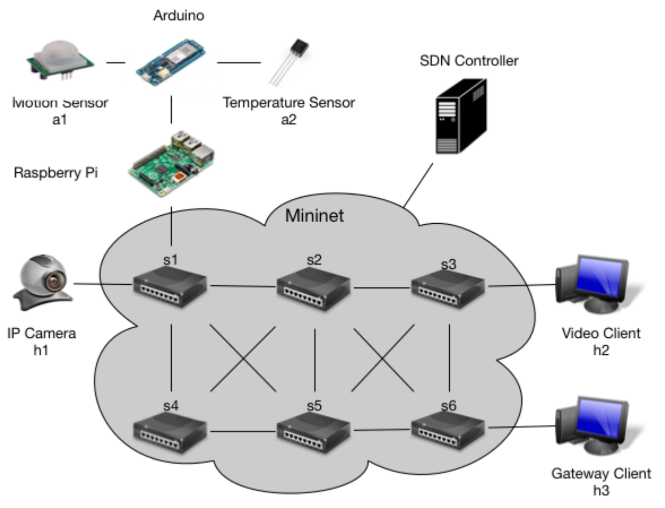

Fig. 3. Network topology of the framework.

of a virtual and a physical network.

The management layer is implemented as a web service, using the Java Spring framework . This ensures a lightweight front-end that is decoupled of the back-end by using REST API. The IoT-QoS Application uses the REST API of Floodlight to insert QoS routes in the network when necessary and it also provides, through REST, the necessary functionalities to the front-end. The front-end is implemented using HTML and jQuery, allowing it to run on any device with browser support. In particular, the front-end can obtain information from the back-end about the existing events, the registered devices and the QoS rules between them.

IV. Evaluation

The proposed framework is evaluated through an experiment on the topology shown in Figure 3. The virtual network consists of six interconnected switches and three virtual hosts: h1, h2 and h3. The Raspberry Pi is connected with one of the virtual SDN switches, enabling it to transmit the sensor data to the hosts within the virtual network. Following the example of the intrusion detection system, an IP camera (h1) transmits footage to the video client (h2). Additionally, a temperature sensor sends data to the gateway client. In the beginning, the quality of the video footage should be low due to certain links in the network being heavily loaded. When the motion sensor detects movement, the framework re-configures the network to ensure high QoS for the video footage, despite network congestion. The ability of the framework to enforce QoS is evaluated by measuring: (1) the throughput of the video footage and (2) latency and packet loss between the temperature sensor and gateway client before and after the motion sensor detects movement. The experiment is successful if the QoS after movement detection improves. Finally, the performance of the framework is evaluated by measuring the time necessary to re-configure the network and enforce QoS once movement is detected.

TABLE II
QoS measurements overview.

	Prior movement detection	After movement detection
Bandwidth (Mbps)	0.5	3.19
Latency (ms)	145.7	12.7
Packet Loss (%)	9	~ 0

A. QoS Measurements

Following the intrusion detection system example, the first communication flow is between the IP camera and the video client, corresponding to hosts h1 and h2 respectively. For that purpose, h1 is configured as a multimedia server running VLC[4]. Host h2 acts as the video client, also running VLC. By default, the multimedia traffic follows the solid line in Figure 4. To improve readability, redundant network links are omitted from the figure. After movement detection, the framework re-configures the network in order to use the QoS path, shown in Figure 4. The throughput of the stream has been measured using Wireshark[5] on host h1. Prior to movement detection, the average throughput is 0.5Mbps. After movement detection, the network is re-configured and the average throughput is 3.19Mbps, demonstrating the ability of the system to adapt.

The other two communication flows in this topology are between the non-IP devices and the gateway client h3. The default and QoS paths used for both flows are indicated in Figure 5. Packet loss is measured at 9% and latency at 145.7ms. After the motion sensor detects movement, the framework re-configures the network to use the QoS paths. The new value for packet loss is measured at $\sim 0\%$. The latency between the temperature sensor and h3 and between the motion sensor and h3 is equal to 10.7ms and 14.7ms respectively, or on average 12.7ms. This completes the experiments, illustrating the effectiveness of the framework to meet the three QoS parameters in the presence of network congestion. An overview of the measured results can be seen in Table II.

B. Performance

The performance is measured in the presence of one, two and three communication flows. For each scenario, ten experiments were conducted in which the re-configuration time is measured once movement is detected. The re-configuration time is the sum of the process time and the communication delay. The former is the time necessary for the framework to take action after the event is detected. The latter is the communication delay between the IoT-QoS Application, SDN Controller and SDN switches. Table III shows the results. In each case, the total amount of time is less than a second, illustrating the capability of the system to timely enforce QoS for the IoT devices.

V. Related Work

Several frameworks have been proposed in research to support QoS using SDN. While these frameworks do not

[4]Described at https://www.videolan.org/
[5]Described at https://www.wireshark.org/

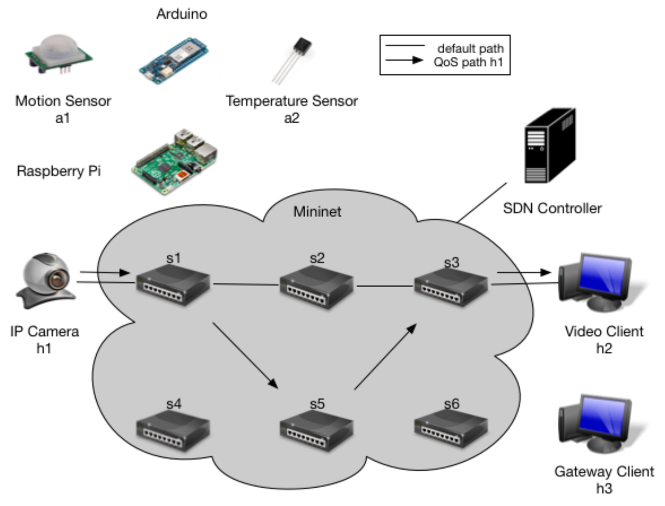

Fig. 4. Default and QoS paths between h1 and h2.

TABLE III
MEASUREMENTS OF RE-CONFIGURATION TIME.

# QoS Rules	Process Time (ms)	Total Time (ms)
1	83.2	100.1
2	125.5	143.8
3	184.5	203.5

support dynamic QoS, they aided the design of the proposed framework. Previous frameworks can be separated into two categories, based on the technique used to enforce QoS: resource reservation and dynamic routing. In the former, network resources are reserved for QoS traffic. In the latter, QoS paths are evaluated dynamically for the communication flows, ensuring the specified QoS guarantees.

In [7], the authors have enhanced the SDN Floodlight controller with modules that enable QoS management. The application is implemented as two Python modules that use the northbound REST APIs that the Floodlight controller provides: QoSManager and QoSPath. The idea is to build queues inside the OpenFlow forwarding devices and use the ”Enqueue” operation provided by the protocol to assign flows to these queues. Each queue provides certain QoS guarantees.

Sharma et al. [9] introduce another resource reservation framework. The authors create a framework, on top of Floodlight, that brings QoS not only within the same Autonomous System (AS), but also across multiple autonomous systems. To achieve this, three default queues are configured on each of the OVS switch ports. The first queue is for control traffic from the controller to the switches, the second for high priority traffic and the third for best effort traffic. Whenever a new flow is discovered, the controller inserts two flow entries in the switch. The first flow entry enforces QoS for packets that belong to this flow by redirecting them to the second queue. The second flow entry is for when best-effort QoS semantics need to be applied to this flow. In that case, the ToS field is disabled and the packets get redirected to the third queue of the outgoing port.

The third framework, presented in [6], uses dynamic routing to support QoS. It calculates a shortest path between two nodes, under constraints and in the presence of multiple flows, through a mathematical model. The framework is written as an SDN application, running on top of Floodlight. The mathematical model is based on the combination of two optimization problems: Multi-Commodity Flow (MCF) and Constrained Shortest Path (CSP). The author combines these two models by using a mathematical model (MCFCSP) that calculates the shortest path for a particular flow within given constraints (CSP), while taking into account the presence of other flows in the network (MCF). Thus, it finds the optimal set of routes within the network.

These frameworks have been designed for traditional services and applications with static QoS requirements. This means that the QoS needs of the services that they support need to be known beforehand and do not change for the duration of the service. The framework proposed in this paper differs from previous work [5], [6], [7], [8], [10] by enabling dynamic QoS support in an IoT environment.

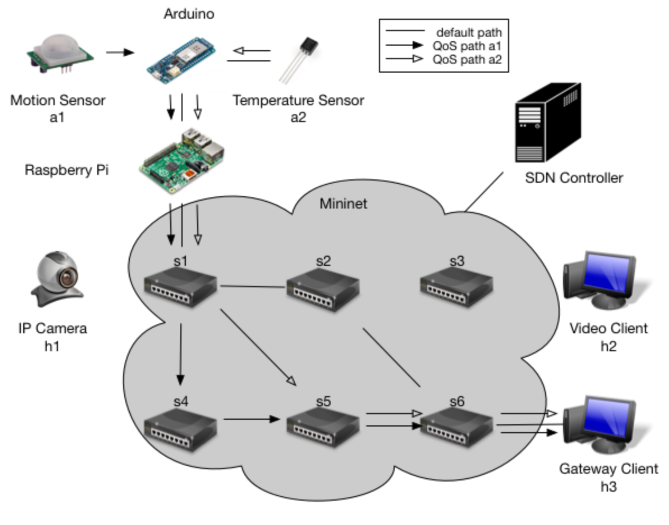

Fig. 5. Default and QoS paths between analog sensors and h3.

VI. CONCLUSION

The Internet of Things is predicted to grow rapidly within the next five years, impacting daily life. For the intended working of several IoT applications, the network needs to provide devices with dynamic QoS guarantees when certain events occur. SDN is a technology that enables network programmability, which is key to tackling the dynamicity of IoT applications. Several frameworks have been proposed in previous research that enhance QoS in computer networks. However, these frameworks focus on the static QoS needs of traditional services, such as multimedia streaming or file transferring.

This paper contributes to the research by proposing a novel framework, on top of SDN, that provides dynamic QoS support for networks in which IoT devices are present. In the field of SDN and IoT, this work is a first step towards treating IoT as a commodity by enabling users to specify QoS guarantees for their IoT devices by introducing representative default QoS profiles, based on existing literature.

This framework is evaluated by performing an experiment on a real-world, hybrid network topology, following the example of an intrusion detection system. The values for QoS parameters are measured for the communication between different devices before and after movement is detected, causing the system to re-configure the network. By comparing the new values of the QoS parameters with the old ones, the effectiveness of the framework is evaluated. Additionally, the performance of the framework is evaluated by measuring the re-configuration time in terms of the amount of QoS rules that are present. The results show that the re-configuration time is in the order of milliseconds and that the QoS parameters have significantly improved after movement is detected.

VII. Future Work

Further research is needed to enable dynamic QoS support for wireless networks. In this paper, SDN was used to enforce QoS within a wired, Ethernet network, as a first step towards providing end-to-end QoS support for the IoT. However, IoT applications are often deployed over infrastructures that are not limited to wired networks, but also contain wireless technologies, such as WiFi and Bluetooth. Ensuring QoS for the IoT on wireless networks requires a different approach, since packets are no longer transmitted over fixed, isolated network links. Wireless SDN is a possible solution, although it is still at its infancy and there are several challenges that remain unanswered, such as handoffs and channel isolation [17].

Finally, the novel framework proposed in this paper limits itself to supporting QoS for IoT within one network. In real world applications, packets often need to travel across the Internet, through different networks that are being managed by different Internet Service Providers (ISP). It is interesting to extend the proposed framework into a distributed solution, consisting of several SDN controllers that would cooperate in order to guarantee the user-specified QoS across the Internet. At first sight, the challenge here is two-fold: on the one hand, SDN is not widely deployed across the Internet and thus a hybrid solution would be necessary that would work on top of both SDN networks and non-SDN networks, ensuring a continuum in the QoS guarantees. On the other hand, additional overhead is produced due to the inter-communication between the different SDN controllers that would manage the Internet as a whole. As with every distributed application, non-functional qualities such as availability, scalability, performance and fault tolerance need to be taken into account, together with their trade-offs.

Acknowledgment

This research is partially funded by the Research Fund KU Leuven and the FWO D3-CPS project. The authors would also like to thank Rafael Bachiller for sharing his experience and wisdom during the course of this research.

References

[1] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[2] M. M. S. I. Altufaili, "Intelligent network bandwidth allocation using SDN (software defined networking)," in *International Journal of Engineering Research and Technology*, vol. 4. ESRSA Publications, 2015.

[3] NEC, "IoT, Meet SDN: How software-defined networking will drive the internet of things," 2014, [Online; accessed 10-December-2016]. [Online]. Available: http://www.nec.com/en/global/ad/insite/article/sdn01.html

[4] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[5] A. Ishimori, F. Farias, I. Furtado, E. Cerqueira, and A. Abelm, "Automatic QoS management on OpenFlow software-defined networks," *API Software Defined Networking Event Demo 2012*, jun 2012.

[6] F. Ongaro, "Enhancing quality of service in software-defined networks," Master's thesis, University of Bologna, 2013 - 2014.

[7] R. Wallner and R. Cannistra, "An SDN approach: quality of service using big switch's floodlight open-source controller," *Proceedings of the Asia-Pacific Advanced Network*, vol. 35, pp. 14–19, 2013.

[8] M. S. Seddiki, M. Shahbaz, S. Donovan, S. Grover, M. Park, N. Feamster, and Y.-Q. Song, "FlowQoS: QoS for the rest of us," in *Proceedings of the third workshop on Hot topics in software defined networking*. ACM, 2014, pp. 207–208.

[9] S. Sharma, D. Staessens, D. Colle, D. Palma, J. Goncalves, R. Figueiredo, D. Morris, M. Pickavet, and P. Demeester, "Implementing quality of service for the software defined networking enabled future internet," in *2014 Third European Workshop on Software Defined Networks*. IEEE, 2014, pp. 49–54.

[10] H. E. Egilmez, S. T. Dane, K. T. Bagci, and A. M. Tekalp, "Openqos: An openflow controller design for multimedia delivery with end-to-end quality of service over software-defined networks," in *Signal & Information Processing Association Annual Summit and Conference (APSIPA ASC), 2012 Asia-Pacific*. IEEE, 2012, pp. 1–8.

[11] C. B. P. Goransson, *Software Defined Networks*. Morgan Kaufmann, jun 2014, chapter 4, How SDN Works.

[12] N. Torabi and M. Hashemi, "Bounded delay transmission of different traffic classes in wireless sensor networks," in *2009 IEEE Wireless Communications and Networking Conference*. IEEE, 2009, pp. 1–6.

[13] I. Technologies, "Internet of Things - Opportunities for device differentiation." [Online; accessed 28-December-2016]. [Online]. Available: https://www.design-reuse.com/articles/36613/internet-of-things-opportunities-for-device-differentiation.html

[14] W. E. Smith, "The truth about digital audio latency." [Online; accessed 28-December-2016]. [Online]. Available: https://www.presonus.com/community/Learn/The-Truth-About-Digital-Audio-Latency

[15] S. Schmid, "Qos based real-time audio streaming in the internet," Ph.D. dissertation, 1999.

[16] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, Irvine, 2000.

[17] C. Chaudet and Y. Haddad, "Wireless software defined networks: Challenges and opportunities," in *2013 IEEE International Conference on Microwaves, Communications, Antennas and Electronic Systems (COMCAS 2013)*, Oct 2013, pp. 1–5.