Toward a Programmable Software-Defined IoT Architecture for Sensor Service Provision On Demand

Thi Minh Chau Nguyen, Doan B. Hoang, Thanh Dat Dang
Faculty of Engineering and IT, University of Technology Sydney, Australia
Emails: {*thiminhchau.nguyen, thanhdat.dang*}@*student.uts.edu.au*, *doan.hoang@uts.edu.au*

Abstract—**In the age of Internet of Things (IoT), sensors form a foundational component of IoT services, yet they are rigid with little capability for programmable configuration or reusability as they are application-specific, manufacturer-specific. Emerging IoT applications often deploy a vast number of sensors which may serve multiple applications. Programmability is thus essential but not found in legacy or current generation sensors. It is challenging to effectively utilize heterogeneity of resources to handle a large number of application demands. Software defined networking and network functions virtualization have proved effective paradigms for provisioning services on-demand and managing network functions and their life cycles. This paper proposes a software defined IoT architecture that captures the spirit of SDN and NFV where a software-defined Internet of Things (SD-IoT) controller can provide services as requested by an application and also manage heterogeneous physical sensors through their virtual representation called software-defined virtual sensor (SD-VSensor) autonomously. In particular, the paper presents the design of a streamline SD-IoT controller, a lightweight and reconfigurable SD-VSensor, and the communication protocol (S-MANAGE) between them. The proposed architecture enables heterogeneous application-specific WSN systems to be recognized and effectively utilized by diverse IoT applications under the orchestration of the SD-IoT controller. Moreover, heterogeneity of sensor nodes or IoT devices can be programmed to achieve sensor services on demand. The preliminary implementation results demonstrate the feasibility and efficiency of the proposed architecture.**

I. Introduction

Wireless sensor networks (WSNs) have become an indispensable sensing data resource for the development of IoT applications. A significant number of WSNs are deployed toward specific demands from various application domains, using different communication protocols and deploying physical devices with different computing and communicating capabilities. The future will witness an increasing number of applications using a vast number of WSN systems over a small area. This leads to inefficient utilization of resources since a deployed system only serves a specific application, while it could be shared by multiple applications with similar interests [1]. However, it is hard to change the behaviors or functions of sensor nodes because their distributed control plane and data plane are both embedded in the device themselves. Many approaches aim to provide common interfaces for development of various applications requiring heterogeneity devices. However, they fail to consider how to manage resource allocation in accordance with the current state of the network usage and how to orchestrate and configure physical sensors to effectively handle application requests on demand. WSNs need a robust architecture that meets these requirements.

To address the problem, we propose a software-defined IoT architecture with three main components, namely a SD-IoT controller; virtual sensors, named SD-VSensors; and an S-MANAGE protocol between the controller and the sensors. The architecture is designed in the spirit of Software-Defined Networking (SDN) and Network Function Virtualization (NFV) principles. SDN paradigm is beneficial to WSNs in terms of energy efficiency, efficient network utilization and services deployments, network programmability and innovation, routing protocol, and cloud integration [2]. It allows the WSN system to be managed and controlled by software autonomously through the programmability of the controller and underlying devices. The SDN architecture comprises of three main planes which are application plane, control plane, and data plane where applications, controllers, and networking devices are accommodated respectively. The controller is both responsible for handling requests from the application and for configuring and managing networking devices in accordance with the application demands. An enabler, called OpenFlow protocol, allows the controller to control and program underlying networking devices. The SD-IoT controller and SD-VSensors are designed in the same spirit. The programmable SD-IoT controller is decoupled from its SD-VSensors (virtual sensors) and is able to manage/configure them autonomously. A SD-VSensor, in turn, can be programmed to represent as well as handle the specifics of its heterogeneous underlying sensors (physical, software, and cluster). Sensor services are decoupled from underlying sensors, which enable the ease of utilizing services and configuring functions of physical sensor nodes. Several benefits can be achieved by this architecture.

- *High interoperability*: diverse IoT applications are deployed over systems of heterogeneous WSN/IoT devices. The architecture provides common interfaces for communication between applications and underlying physical devices.
- *Efficient data collection*: a majority of IoT applications demand rapid access to sensed data. Their expectation is to effectively and correctly obtain the required data.
- *Programmability*: Emerging IoT applications are to be provisioned on-demand and rapidly. This

necessitates a dynamically programmable IoT infrastructure.

- *Energy efficiency*: Taking into account the energy concern in designing a WSN architecture [3], constrained physical sensor nodes and IoT devices may only collect and transmit their data, leaving computing and routing functionalities to purpose-built- devices.

The remaining of the paper is organized as follows. Section II presents related work. Section III presents the proposed architecture. Section IV describes the implementation. Section V presents preliminary performance evaluation results. Section VI concludes the paper and presents future work.

II. Related work

Efforts have been made to extend the SDN principles to WSNs [2], but they do not pay much attention to the controller, the sensors, or the API between them. Most try to reuse the OpenFlow with minor modifications and with little consideration of the WSN environment.

Several attempts have tried to combine SDN and virtualization techniques into the IoT architecture. In [4], a SDN-based IoT platform is proposed with four layers which are a service layer accommodating IoT services, a computing layer including SDN controllers and billing functions, a communication layer consisting of routers and gateways for forwarding data, and a device layer including sensors for collecting data. Nevertheless, with the architecture, the controller cannot program wireless system including sensing devices and sink nodes according to an application on demand. In [5], a SDN-based framework with NFV implementation is designed, but the framework mainly discusses the importance of distributed OS in the control plane without concern for IoT device management and programmability. Another software-defined industrial IoT platform is proposed to allow legacy shop floor equipment to exchange data with cloud-based manufacturing applications [6]. It is designed as an SDN controller and is attached to a machine which needs to connect to the cloud. The platform can connect any type of physical machine to the cloud, but the approach only enables a machine to serve one application. Moreover, it does not discuss how to manage legacy devices with the OpenFlow protocol to enhance the controller's device management.

Virtual sensors have been defined in a number of applications. In [7], it is defined as an abstraction providing an aggregated data computed from different sensing data. It can also supply a predicted data in the event a physical sensor failed. It can be used to share some complex tasks with the physical sensor nodes by enhancing virtual sensors with different functions [8]. One interesting design of virtual sensor which is provided by middleware architecture is proposed in SenseWrap [9]. The architecture allows the virtual sensor to represent a physical sensor and its sensing reading can be used by multiple applications. However, its difficulty is how to route returned data to a correct destination. Furthermore, the middleware was designed before SDN became established; hence it is difficult to integrate the middleware platform with SDN networks.

III. The proposed SD-IoT architecture

To gain the benefits of the SDN paradigm, the proposed SD-IoT architecture is also designed with three main planes as shown in Fig. 1; 1) the application plane for accommodating IoT applications; 2) the control plane for processing application demands and configuring resources within the data plane according to the requirements; and 3) the data plane accommodating virtual and underlying sensors that operate under the control of the controller.

The application plane allows developers to deploy their applications without concern about the complexity of the underlying system by providing them with its abstraction. The application requirements are translated and communicate with the controller through a northbound interface.

The control plane includes a SD-IoT controller which is a manager and orchestrator of a local WSN/IoT system. The controller communicates with the application through a northbound interface (NBI) and with the SD-VSensors through a southbound interface (SBI). In this plane, we introduce a SD-IoT controller to both handle application demands and manage underlying systems. A proposed SBI, called S-MANAGE protocol, between the SD-IoT controller and the SD-VSensor, enables the controller to update the status of underlying WSNs/IoT systems and configure them according to the application requests and taking into account the availability of the underlying resources.

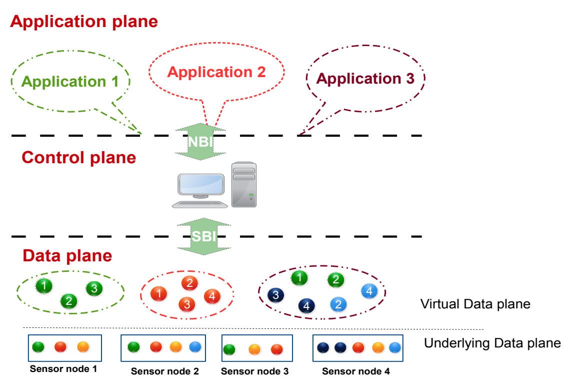

Fig. 1: SD-IoT architecture

The data plane includes two sub-planes, called virtual and physical data plane. The virtual data plane is comprised of virtual sensors (SD-VSensors), which are representatives of physical/software sensor nodes. The SD-VSensors are smart and programmable representations/interfaces of their underlying sensors which may include software sensors, physical sensors, and physical actuators. Via the SD-VSensor, IoT applications can request for the desired sensor services without concerning with the peculiarities of physical sensors or their various communication protocols. For example, they can request for sensing data or controlling actuators of physical sensor nodes, for turning physical sensors on/off, or for adjusting the frequency of getting sensing data via their virtual representatives. The physical data plane includes heterogeneous physical sensors/ IoT devices that have various communicating and computing capabilities. Details of main components in the proposed architecture are shown in Fig. 2.

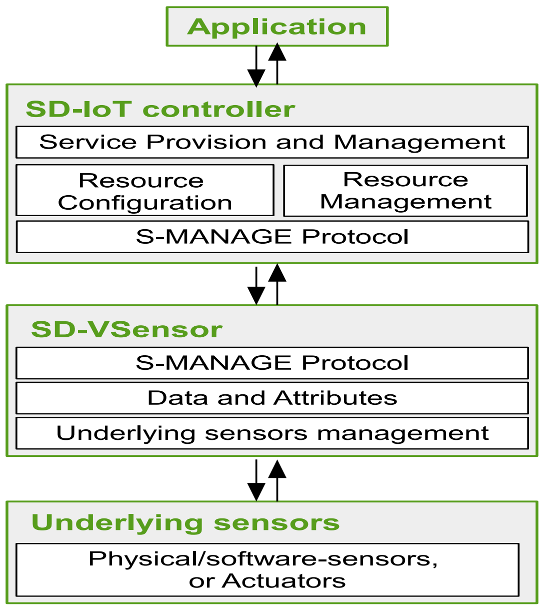

Fig. 2: Functional components of the proposed SD-IoT architecture

A. *Software-defined IoT controller*

The SD-IoT controller processes application requests and orchestrates its resource according to the requests. To this purpose, it uses the northbound interface (NBI) and the southbound interface (SBI).

The NBI makes it easier for application developers to express their demands based on the abstraction of the complex underlying systems. The controller uses a library of various application communication protocols to be able to communicate with diverse applications and translates the requirements into commands which are applied to virtual sensors to provide sensor services.

Via the SBI, the controller can control, manage, and configure the virtual sensors and through which configure underlying devices. To this purpose, the S-MANGE protocol is defined as a communication protocol between the SD-IoT controller and the SD-VSensor. It enables the controller to utilize services provided by each SD-VSensor and configure functions of the virtual sensor.

The controller includes four main functional elements.

Service provision and management has a responsibility for providing sensor services for the applications and managing application-service source connections.

- *Service provision* i) advertises its services to the application with the connection information regarding domain name, IP address of the controller, and its services; ii) listens to interests from applications; and iii) looks up its service lists and provides the application with appropriate resources requirement.
- *Service management* *i*) maps application requests to potential SD-VSensors; *ii*) provides TCP/IP communication channel for a SD-VSensor and its corresponding application; *iii*) updates and stores an Application_SD-VSensor mapping table and their corresponding services.

Resource management discovers service resources from SD-VSensors; configures an identification for the SD-VSensor and its service list; and provides the list for the Service Management. Particularly, it listens to SD-VSensors' advertisements about its sensor services and updates the service lists with information concerning SD-VSensor's identification and service specifications including sensing service name, sensing service ID, actuating ID or name.

Resource configuation manages the availability of SD-VSensors and their sensor services based on the Application_SD-VSensor mapping table and sensor service lists provided by the service management, and the resource management respectively. Using the information, it creates forwarding rules for matching data between the application and its required SD-VSensors. Moreover, it maintains a flow table including the forwarding rules for matching data between the application and the corresponding SD-VSensor. In addition, it updates the rules associated with the orchestration of the underlying resources using S-MANAGE messages.

The S-MANAGE protocol defines communicating message types between the controller and SD-VSensors. The controller uses the S-MANAGE messages to get sensor services from the SD-VSensor; configure a connection between the application and the SD-VSensor; configure SD-VSensor's functions and behaviours; and update SD-VSensor status when it is on/off.

B. *Software-defined virtual sensor (SD-VSensor)*

1) SD-VSensor as the interface for underlying sensors: As mentioned earlier, the proposed virtual sensor is named SD-VSensor. It can represent one or more physical sensor nodes; one or more software sensor nodes (software implementation of sensed data functions such as averaging, aggregating, etc.), or one or more actuators. We denote the sensor node(s) represented by SD-VSensor by the underlying sensor(s). The SD-VSensor is able to communicate with the SD-IoT controller via a special protocol, called S-MANAGE protocol, and is programmable via the S-MANAGE messages.

The virtual sensor may possess a driver of the underlying sensor, so they can communicate with each other via the communication protocol of the physical sensor node. The SD-VSensor is a software module which enables a physical/software sensor node to connect to an IoT application. It is responsible for communicating with others on behalf of its underlying sensors regardless of which communication protocol used by the application. It receives requests from the controller or applications and translates them to understandable commands for the underlying sensors. The underlying sensors are responsible for getting data and forwarding the data as dictated by the SD-VSensor.

2) SD-VSensor's services: A SD-VSensor represents its underlying sensors hence its sensor services are the services provided by the underlying sensors. The SD-VSensor knows the sensing capabilities/services of its underlying sensors either through direct attachment or through an advertising/discovering process between the underlying sensors and the SD-VSensor. A SD-VSensor can be located at its underlying sensor or at a SD-IoT controller. It always registers the services to its controller.

The underlying sensors can provide services such as sensing, actuating, processing (aggregating, or routing), or communicating. Regarding sensing services, the underlying sensors may have more than one sensing unit, so they can provide multiple types of sensing data or called sensing services. They can also aggregate several sensing readings to produce a new sensing data.

C. S-MANAGE protocol

The S-MANAGE protocol is used to establish a secure communication channel between SD-IoT controller and SD-VSensors for control and management purposes. It enables the controller to i) acquire sensor services from the underlying sensors via the SD-VSensor; ii) control the behavior of the SD-VSensor; and iii) manage and configure underlying sensing/actuating services indirectly via the SD-VSensor. Moreover, by using S-MANAGE messages, the controller can set up the connection between an application and a SD-VSensor. There are two types of S-MANAGE messages.

- *SD-IoT Controller to SD-VSensor*: by using this message type, the SD-IoT controller can discover sensor services and configure SD-VSensor. The controller can set identification and a rule table for a SD-VSensor. It also sets up a communication channel for the Application and the appropriate SD-VSensor. Moreover, the controller can request services of the SD-VSensor and through which control the actuators of physical sensors.
- *SD-VSensor to SD-IoT Controller*: this message type enables the SD-VSensor to register its services to the controller; reports its status of being on/off; notifies its status about the connection to the application in case it directly communicates with the application without via the controller.

IV. Implementation

The main aim of the proposed SD-IoT architecture is to facilitate the provision of sensor/IoT services on-demand. That requires the controller to understand the application/service requirements and to possess the ability to muster, construct, and program its sensing resources to provide the needed services. This, in turn, necessitates the programmability of its resources. A SD-VSensor thus does not only possess the capability of being programmed on demand but also capable of configuring its underlying sensors. On implementing the SD-IoT architecture, we are inspired by an Active Data-Centric framework (ADC) [10] for data protection in a cloud environment in which data is encapsulated by an intelligent shell to make it an active data unit that can communicate its functionality and properties with its supervisor for active protection. Our SD-VSensor will take a role similar to an active data unit whose data is provided by its underlying sensors and our SD-IoT controller will expand the role of a supervisor beyond its protection capability. Our preliminary implementation is as follows.

A. SD-IoT controller implementation

The SD-IoT controller is designed to manage and configure one or more SD-VSensors. The controller has four main modules. An application interface is defined for the communication between the applications and the SD-IoT controller regarding application requests and return results. A monitor is used for listening to an application request, mapping it to the SD-Controller's sensing service list, and then locating an appropriate SD-VSensor. A trigger is used for activating the needed SD-VSensor. A communicator (handling S-MANAGE) is used for establishing connections between the SD-IoT controller with one or more SD-VSensors. (Additional features concerning security aspects will be implemented in the future). The main functional modules are written in Java.

B. SD-VSensor implementation

The SD-VSensor has two main components: a Shell and a Core. The Shell houses software for communication with both the SD-IoT controller and the underlying sensors. The Shell handles the S-MANAGE protocol on the controller facing side and sensor-specific protocols on the underlying sensor facing side. The Core houses attributes for performing its functionality including SD-VSensor_ID, Sensing-Service_ID, Sensing Service_Name, Actuating_Service_ID, Actuating Service_Name, Physical Sensor node address, Application-VSensor mapping table. It also has a data buffer to store (temporarily) sensing data from its underlying sensors (e.g., Raspberry Pi) and transfers it to a destination directed by the controller. An executable agent (in fact, a virtual function (VF) in the network functions virtualization context) is used for running the SD-VSensor.

To demonstrate the underlying sensor management functions for communicating with the Raspberry Pi, a communication channel is defined between the SD-VSensor and the Raspberry Pi. It is established by using the IP address and port number of the two entities. The SD-VSensor uses it to request sensing service stored in its associated Raspberry Pi.

C. Sensing service provision workflow

An application sends a sensing service request to a controller. The controller processes the request and triggers a proper SD-VSensor. The requested SD-VSensor communicates with its corresponding Raspberry Pi to obtain the required service and only the SD-VSensor can communicate with the Raspberry Pi. To respond the request, the Raspberry Pi returns required data to its represented virtual sensor. The SD-VSensor returns the

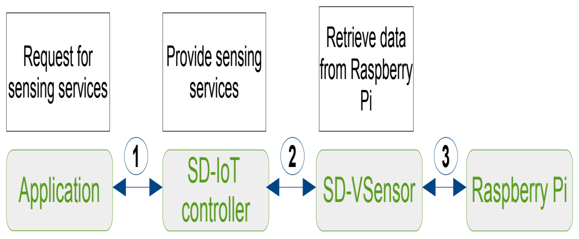

Fig. 3: Sensing service provision workflow

result to a destination specified by the controller. Figure 3 depicts the sensing service workflow.

V. Performance Evaluation Results

A. Experimentation setup

The experimental environment has been set up as follows: the controller is deployed in a laptop Dell Alienware equipped with Intel(R) Core(TM) i7-4900MX CPU, 3.5 GHz and 32GB of RAM, running Windows 7, 64 bits. The controller is implemented using Java 7. Requests are accessed via RMI interface. We use MySQL to store virtual sensors' information within the controller. The SD-VSensors are implemented at the controller. Four Raspberry Pi 3 devices are used as the underlying devices. These devices are implemented with the Raspbian release 1.3 images. The SD-VSensor connects to the Raspberry Pi via a Wifi network.

B. Performance Evaluation

This implementation is conducted with some assumptions. A Raspberry Pi is represented by a SD-VSensor. A Raspberry Pi has one, two, five, or ten underlying sensors, so a SD-VSensor can provide one, two, five, or ten different sensing data types referred as sensing services. The data readings are formatted in XML files with the same size and stored in each Raspberry Pi. The returned data is sent from Raspberry Pi to the application. The number of requests generated by the application can be varied from 1 to 200. All results are an average of five trials. There are six testing scenarios.

This work firstly investigates new components for the SD-IoT architecture, so to evaluate its efficiency, we examine the processing time (in milliseconds) of a sensing service provision. The processing time is counted from the time when an application request is sent out until the application gets the result. It is composed of four elements: 1) t1: time for the application sending a request to the controller, and the controller processing the request and then triggering an appropriate SD-VSensor to handle the request; 2) t2: time for the SD-VSensor running and sending the demand to its corresponding Raspberry Pi; 3) t3: time for the Raspberry pi returning the proper sensing service to the SD-VSensor; and 4) t4: time for the SD-VSensor sends the returned data to the application. We examine the processing time over six test cases. In each case, the number of requests, or the number of sensing services per request, or the number of SD-VSensor varies. In particular, we generate one or multiple requests for one or more sensing services which are provided by one or more SD-VSensors. The first four cases are for testing the operation of the controller and one SD-VSensor, while the other two cases for testing the operation of the controller and multiple SD-VSensors. The application initiates the request.

1) Case 1: one request - one sensing service - one SD-VSensor: In this case, an application sends a request for one sensing service provided by one SD-VSensor. The result shows that the average processing time for such request is about 2.67ms. The average value is compared to that of the following cases for performance evaluation of the architecture.

2) Case 2: one request - multiple sensing services - one SD-VSensor: In this case, an application demands multiple types of sensing services from one SD-VSensor. For each request, we increase the number of sensing services from 1 to 2, 5, or 10. The experimental results in Fig. 4 show that the average processing time of each request increases due to the time needed for processing the request for different kinds of sensing service and also for the SD-VSensor's receiving multiple responses from its associated Raspberry Pi. However, the request for a higher number of sensing services can achieve a single sensing service with less processing time than that is obtained from the request for less number of sensing services. This can be seen by dividing the total processing time to the number of demanded sensing services per request. Particularly, the average processing time for obtaining one sensing service is about 1.1ms for the request for 10 sensing services while it is about 2.7ms for the request for one sensing service. The reason is the saving time regarding the SD-VSensor's only sending the whole required sensing services once it completely collects the needed data from its Raspberry Pi instead of sending separated results to the application.

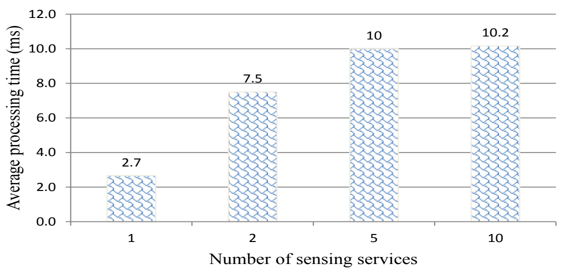

Fig. 4: Average processing time for one request for multiple sensing service from one SD-VSensor

3) Case 3: multiple requests - one sensing service - one SD-VSensor: In this case, many applications are interested in one sensing service type from one SD-VSensor. In Fig. 5, the average processing time of each request is ranging from 2.5 to 3.2ms which is slightly higher than the average processing time for one request in case 1 because of the necessary time for the controller's processing multiple requests. Meanwhile, the average time for processing each request per multiple requests becomes less as the number of requests is higher. The reason is that the SD-VSensor can save time in obtaining the sensing service requested by multiple applications by sending one request to the Raspberry Pi for such sensing service and returning the result to multiple destinations.

4) Case 4: multiple requests - multiple sensing services - one SD-VSensor: In this case, many requests for different sensing services are sent to one SD-VSensor. Different from case 2 in terms of the number of requests, in this case, we generate multiple requests for 2, 5, or 10 sensing services. The number of requests is 10, 50, 100, 150 and then 200. Overall, the request for more sensing service needs more processing time. The average processing becomes higher as the number of request increases since the consuming time by the con-

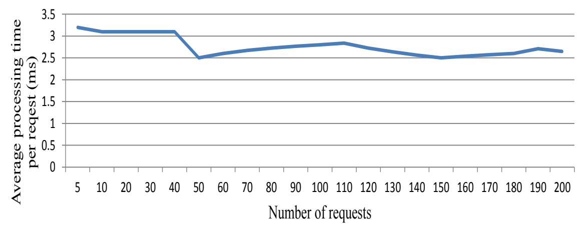

Fig. 5: Average processing time of each request per multiple requests for one sensing service from one SD-VSensor

troller's processing requests and indicating the right SD-VSensor. However, by dividing the average processing to the number of requests, as shown in the Fig. 6, the average processing time of a single request per multiple requests for even 2, 5, or 10 sensing services is slightly different and fluctuates between 2.5ms to 3.8ms. Obviously, the request for more sensing services needs more processing time, but they are much less than the corresponding values in case 2. This is because the processing time is saved when one request can be used for requiring multiple sensing services and the SD-VSensor can reuse of the results for similar interests and save time for getting the required sensing services from the Raspberry Pi.

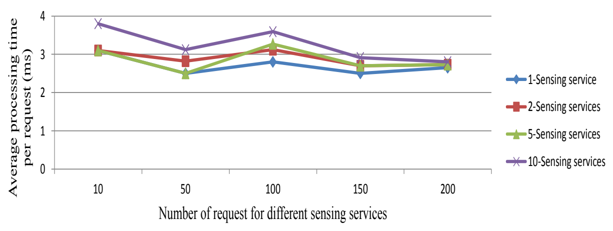

Fig. 6: Average processing time for each request per multiple requests for different sensing services from one SD-VSensor

5) Case 5: one request - one sensing service - multiple SD-VSensors: In this test, an application requests for one type of sensing service provided by four SD-VSensors. This case costs more processing time (about 6.2ms) than that in case 1 (around 2.7ms) since the controller has to process more SD-VSensors in its resource list. The processing time for obtaining the sensing service from two SD-VSensors is about 12.4ms, twice of that from one SD-VSensor, whereas the difference becomes smaller for three and four SD-VSensors, about 15.4ms and 15.6ms respectively. By dividing the total processing time to the number of SD-VSensors, the processing time of each SD-VSensor is less as the number of required SD-VSensor rises. It can be concluded that the increase in the number of SD-VSensor results in a slight rise in the average processing time.

6) Case 6: multiple requests - one sensing service - multiple SD-VSensors: In this test, we increase the number of requests from 1 (as in case 5) to 5, 10, 50, 100, 150, and then 200. As shown in Fig. 7, the processing time for these requests increases as the rise in either the number of requests or the number of SD-VSensors. However, the growth is in a linear, not an exponential manner. The increase is from the controller's processing multiple requests and multiple SD-VSensors' returning results to multiple destinations.

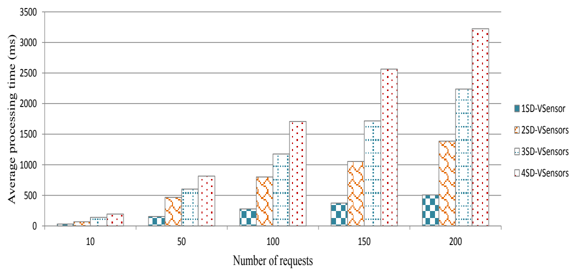

Fig. 7: Average processing time for multiple requests for one sensing service from one or more SD-VSensors

VI. Conclusion and Future Work

In this paper, we present an efficient SD-IoT architecture whereby the SD-VSensor provides an intelligent and programmable interface to the underlying physical and software sensors/IoTs and their sensing services. Through the management of the SD-VSensor underlying sensors/sensor networks can be utilized by multiple applications under the orchestration of the SD-IoT controller. The initial implementation results show that the proposed architecture can efficiently provide multiple sensing services for multiple applications simultaneously. Our future work aims to present detail specification of each component in the architecture and a full implementation to demonstrate the capability of proposed architecture in SDN environment.

References

[1] A. Zanella and et.al, "Internet of Things for Smart Cities", *IEEE Internet of Things Journal*, vol. 1, pp. 22-32, 2014.
[2] T. M. C. Nguyen, D. B. Hoang, and Z. Chaczko,"Can SDN Technology Be Transported to Software-Defined WSN/IoT?," in *IEEE Int. Conference on Internet of Things and IEEE Green Computing and Comm. and IEEE Cyber, Physical and Social Computing and IEEE Smart Data*, 2016, pp. 234-239
[3] N. Kamyabpour and D. B. Hoang, "A Hierarchy Energy Driven Architecture for Wireless Sensor Networks," in *2010 IEEE 24th Int. Conference on Advanced Information Networking and Applications Workshops*, 2010, pp. 668-673.
[4] Y. Li and et.al, "A SDN-based architecture for horizontal Internet of Things services," in *2016 IEEE International Conference on Communications (ICC)*, 2016, pp. 1-7.
[5] J. Li, E. Altman, and C. Touati, "A General SDN-based IoT Framework with NVF Implementation," *ZTE Communications*, vol. 13, pp. 42-45, 2015-09 2015.
[6] A. Bahga and et.al, "Software Defined Things in Manufacturing Networks," *Journal of Software Engineering and Applications*, vol. 9, p. 425, 2016.
[7] S. Kabadayi, A. Pridgen, and C. Julien, "Virtual sensors: Abstracting data from physical sensors," in *Proceedings of the 2006 International Symposium on on World of Wireless, Mobile and Multimedia Networks*, 2006.
[8] A. Gupta and N. Mukherjee, "Implementation of virtual sensors for building a sensor-cloud environment," in *8th International Conference on Communication Systems and Networks (COMSNETS)*, 2016, pp. 1-8.
[9] P. Evensen and H. Meling, "SenseWrap: A service oriented middleware with sensor virtualization and self-configuration," in *International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, 2009, pp. 261-266.
[10] L. Chen and D. B. Hoang, "Active data-centric framework for data protection in cloud environment," in *Proceedings of the 23rd Australasian Conference on Information Systems*, 2012, pp. 1-11.