S-MANAGE Protocol For Software-Defined IoT

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

Abstract—**The Internet of Things (IoT) has started to make a real impact with many IoT-based services in agriculture, smart farming, smart cities, personal health, and critical infrastructures. Sensor/IoT devices form one of the indispensable elements in these IoT systems and services. An effective IoT system requires the interoperability among its heterogeneous physical devices, but this presents a significant challenge regarding various communication protocols, networking management policies, as well as data processing approaches. Software-defined paradigm is considered essential for managing and provisioning IoT services on demand. An emerging solution is the application of software-defined networking (SDN) and Network Function Virtualization (NFV) in programming WSN/IoT systems. However, these technologies cannot be directly deployed due to the differences in the functionality of SDN network devices and sensor/IoT devices as well as the limitation of resources in IoT devices. We proposed the software-defined IoT(SD-IoT) model in our earlier work. This paper focuses on the S-MANAGE protocol that enables an SD-IoT controller to control and manage sensor/IoT devices via their virtual representation, called software-defined virtual sensors (SDVS). The paper presents in detail the design and the implementation of the S-MANAGE southbound protocol.**

Keywords - S-MANAGE protocol, Software-defined virtual sensor (SDVS), Software-defined IoT (SD-IoT)

I. Introduction

An Internet of Things (IoT) environment accommodates numerous IoT devices (we use IoT devices to include networked sensors in this paper) with various sensing, computing, communicating, and actuating capabilities. Deployment of an IoT application becomes challenging owing to the large coverage of the application, the limitation of resources of IoT devices, and the heterogeneity of the environment [1]. IoT applications often overlay and share the deployments of IoT devices, and this presents difficulties and challenges in the interaction and sharing of information between the devices and the applications [1]. The concern is on the programmability of various sensor nodes in response to demands from diverse IoT applications. Among programmable solutions to the programmability of Wireless Sensor Networking/Internet of Things (WSN/IoT) systems, various SDN-based solutions are proposed [2]. However, challenges remain when applying the software-defined networking (SDN) paradigm to the constrained WSNs/IoTs [3].

The fundamental element of an IoT system is the underlying resources which provide necessary data for IoT applications. Therefore, an IoT architecture necessitates to not only control and manage the underlying resources but also orchestrate them to satisfy application demands. However, architectural solutions for provisioning various IoT applications are still immature. A majority of the proposed approaches are vertically integrated, so it is difficult for the infrastructure to handle various IoT application demands that require horizontal capabilities from other subsystems. Many attempts have been made to address IoT platform architectures and applications on demand, challenging issues remain and include scalable and dynamic resource discovery and composition, context-awareness, integration of intelligence, interoperability, reliability, security and privacy, and system-wide scalability [4]. Thus, we propose a software-defined Internet of Things model that adopts SDN and NFV principles and adapt and deploy these technologies to IoT devices and IoT applications.

The software-defined networking approach addresses many existing problems concerning network management and provisioning resources required by network services. The SDN principle separates the network control plane from the data plane of networking devices and allows the provision of services on demand through a programmable and logically centralized controller. Autonomous management of network devices is enabled under SDN. SDN architecture comprises three main planes which are application plane, control plane, and data plane. The control plane centrally controls and manages the behavior of the whole network existing in the data plane via a Southbound Interface (SBI). To provide network services to the application plane, it uses a Northbound Interface (NBI) to expose the abstraction of the underlying network.

However, it is not feasible to completely and directly apply SDN techniques to the resource-limited WSN/IoT environment owing to its constraints [5]. In this paper Network Function Virtualization (NFV) is deployed to realize software-defined virtual sensors as a representation of the underlying sensor/IoT resources. This technology is thus applied readily to WSN/IoT environment for creating a virtual representation of sensors/IoT devices that are utilized by multiple IoT applications simultaneously. This virtual representation offers a solution to enrich the features of limited sensors/IoT devices. By applying both SDN and NFV principles to the proposed software-defined IoT model, diverse underlying sensor nodes can be programmed in accordance to the IoT application requests.

In earlier work, we proposed a software-defined Internet of Things (SD-IoT) model for IoT clusters where the S-MANGE protocol was proposed as a communication bridge between the SD-IoT controller and the software-defined virtual sensor/IoT (SDVS). The controller sets up and configures the SDVS by using the S-MANAGE which is designed to deal with constraints of IoT systems. This paper investigates the design and the implementation of S-MANAGE. S-MANAGE is designed to both configure SDVSs and control the behaviour of the underlying networked IoT resources.

With the proposed SD-IoT model, we can achieve three main aims: i) controlling and managing IoT infrastructures in according to IoT application demands; ii)

enabling heterogeneous IoT devices to integrate to the SD-IoT system, and iii) integrating SD-IoT into wider SDN domains where cloud-based IoT applications reside in order to take advantages of IoT resources.

The remaining of the paper is organized as follows. Section II reviews related works. Section III describes the overall SD-IoT architecture. Section IV presents the design and the specification of the S-MANAGE protocol in terms of packet format, packet type, forwarding table, and configuring table. Section V describes the implementation prototype and evaluates its performance. Section VI concludes the paper.

II. Related work

With regard to the communication interface between the controller and underlying devices, existing proposals mainly suggest modification without actual implementation, for example, Sensor OpenFlow [3], and SDWN [6]. Only the SDN-WISE [7] proposal provides details of its design and implementation performance. Detail of these proposals is discussed in our previous work [8].

[9] applies the OpenFlow protocol in their proposed flow-sensor. The flow-sensor communicates with the access point via the OpenFlow protocol. Their implementation results demonstrate how OpenFlow benefits controlling and monitoring sensor traffic flow, but there is no effort in making the OpenFlow protocol suitable for constrained sensor nodes.

Regarding a complete design of a software-defined wireless sensor network, there are two major efforts, the proposed SD-WISE [10] using the SDN-WISE [7], and soft-WSN [11]. The SDN-WISE protocol mainly focuses on programming forwarding behavior of a sensor node. The main components of the protocol are described, and the design is evaluated via a real implementation. However, its main aim is to program a nodes forwarding behavior without the concern about programming a node's functionality.

Meanwhile, the soft-WSN architecture is proposed to provide IoT application-aware services. Its structure also includes three players, application, control, and infrastructure. A controller is located in the control layer to manage and control the network and devices in the infrastructure. The controller is designed with two management policies, topology and device management. The main focus is on the design of the controller and the sensor node architecture. The communication between the two entities is based on traditional protocols, IEEE 802.15.4 and IEEE 802.11. There is no effort on solving the complexity of deployment of flow-tables to the sensor node.

III. SD-IoT Model

To gain the benefits of the SDN principles, the SD-IoT model is also structured in three layers: application, control, and data as depicted in Fig. 1.

The application layer is where developers can deploy their IoT applications without the knowledge of the underlying IoT infrastructure.

The control layer accommodates the SD-IoT controller. It is a bridge between the application layer and the data layer. It provides the application layer with a global view of the underlying resources as well as an efficient interface to express their interests on the underlying IoT resources. Meanwhile, it provides the underlying resources with an interface to update their status, attributes, as well as sensor services. With the knowledge of both requirements and capabilities of the IoT resources, it can provide sensor services for IoT application on demands.

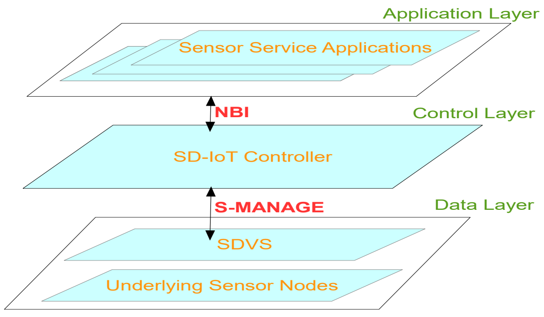


Fig. 1: SD-IoT Architecture

The data layer hosts IoT devices or IoT infrastructure. Different from the SDN data layer, the SD-IoT data layer is designed with two sub-layers, called virtual and physical data layers. The virtual data layer is proposed as an interface between the SD-IoT controller and the physical devices. The virtual data layer enables the controller to manage and control the underlying resource in the physical data layer.

A. SD-IoT controller

The SD-IoT controller is responsible for i) processing application requests, ii) orchestrating resources, iii) updating knowledge of the underlying resources, and iv) controlling and managing the underlying resources according to IoT application demands.

To handle these responsibilities, the controller houses several core modules: application handler, resources manager and orchestrator, configuration manager, and a database for storing information concerning the underlying resources.

Particularly, the SD-IoT controller makes it possible for the application layer to specify their demands via a northbound interface (NBI). The controller interprets these requirements into the SD-IoT resource specific language in order to orchestrate the resources to meet the IoT applications requirements. To configure the underlying resources, the controller makes use of a southbound interface (SBI) defining resource-specific messages to configure these devices.

B. S-MANAGE protocol

The S-MANAGE protocol is proposed as a southbound interface between the SD-IoT controller and the virtual data layer. Via the SBI, the controller can manage and configure software-defined virtual sensors (SDVS) in this layer.

It defines the message structure, and message types exchanged between the controller and the virtual layer. These messages are for configuration management and control of behaviours of SDVSs.

C. Software-defined virtual sensor (SDVS)

The software-defined virtual sensors (SDVS) are defined as representatives of physical/software sensor nodes or IoT devices locating in the physical layer. It is configured with core features and attributes of a physical sensor node, and software-defined function

(SDF). The core modules enable the SDVS behaves as the represented physical IoT device, so it can interact with the represented devices via its communication-specific protocol. Furthermore, the SDF allows the controller to enhance the SDVS with processing, computing, or forwarding functions. Particularly, the forwarding and configuring function is implemented to the SDVS as SDF. The SDVS is supposed to be connected to its underlying IoT device and is able to act like the represented IoT device. In addition, it is installed with S-MANAGE protocol features, so it can communicate with the controller.

IV. S-MANAGE PROTOCOL

In traditional IP networks, routers are used to relay packets (or datagrams) according to its lookup table determined by a routing protocol. Packets are treated as independent elements not related to other packets that may belong to the same source-destination connection. Traffic flow is normally associated with packets belonging to an end-to-end TCP connection. In order to completely specify a flow at a router, a large number of identifiers are needed from transport layer ID, network layer IP, VLAN ID, MAC layer ID, router port IP are needed. As a consequence, a flow in OpenFlow protocol requires some 12 matching parameters to identify. Clearly, this is not needed in sensor/IoT networks where the end devices are not routing devices in the traditional network. Many devices do not use TCP/IP protocol, direct deployment of OpenFlow in WSN/IoT networks is not appropriate. Furthermore, OpenFlow SDN network still requires OF-CONFIG or other protocols for device configuration. What we need is a streamline protocol in WSN/IoT networks that can handle both configuration of the IoT devices and simple type of sensed data but in the same spirit as flow in OpenFlow. S-MANAGE protocol is proposed to do just that. As the southbound protocol, the S-MANAGE protocol is proposed as a southbound interface between the SD-IoT controller and the virtual data layer. Via the southbound interface (SBI), the controller can both manage and configure software-defined virtual sensors (SDVS) in this layer.

The S-MANAGE protocol is for managing and programming the SDVS within the virtual data layer and indirectly via which to configure their represented physical devices. S-MANAGE makes it possible for the controller to program sensors or IoT devices not only their forwarding behaviors but also other functionality.

The protocol is proposed according to the spirit of two protocols, OpenFlow [12] and OF-CONFIG [13]. The OpenFlow focuses on flow rules as setting, modification, deletion, or adding rules for controlling forwarding behavior of OpenFlow switches. Meanwhile, the OF-CONFIG enables configuring an OpenFlow Switch itself as the setting of port number, IP address, or interfaces.

The protocol enables the management and configuration of representations of sensors/IoT devices to be based on two proposed instruction tables, called forwarding and configuring table. The forwarding table instructs a SDVS to handle an arriving packet, while the configuring table guides a SDVS to configure its represented underlying nodes.

The protocol allows the controller to $i)$ install instruction tables on a SDVS for configuration purposes, $ii)$ get information concerning the SDVS's features, functions, the and the status of its underlying sensors, and $iii)$ collect statistics associated to the SDVS's operation as the number of processed packets or required sensor services from IoT applications.

In addition, via the protocol, a SDVS is able to $i)$ update the controller with their status and attributes, and $ii)$ ask for instructions on processing an incoming packet or configuring its underlying sensors or IoT devices.

The S-MANAGE defines communication methods between the controller and a SDVS. It specifies exchanged message types between the two entities, the message format, the structure of instruction tables, and how a SDVS is programmed and operates based on these tables' instructions. Details of the protocol design are described in the following sections.

A. *S-MANAGE packet header*

The structure of the S-MANGE packet comprises a header and a payload. All S-MANAGE messages begin with a S-MANAGE header as depicted in Fig. 2. The header size is 10 bytes. It includes the following parts.

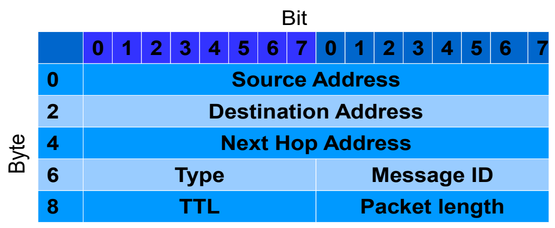


Fig. 2: S-MANAGE packet header.

- Source Address (2 bytes) is an address of a source sending a packet.
- Destination Address (2 bytes) is a destination address of a packet.
- Next hop address (2 bytes) is an address of a hop in the list providing the path of a packet from the source to the destination.
- Type (1 byte) indicates a packet type.
- Packet length (1 byte) indicates the length of a packet including its header and payload.
- TTL (1 byte) is time to live of a packet. It is reduced by one at each hop.
- Message ID (1byte) is an identification of a packet.

The payload carries the content of a packet. Different types of packets carry different kind of information which represents different purposes of a sender. Therefore, we define the following S-MANAGE message types to achieve the expected purposes.

B. *Message types*

The S-MANAGE message types are grouped into three categories, $i)$ controller to SDVS, $ii)$ asynchronous (SDVS to controller) and $iii)$ symmetric (controller/SDVS to SDVS/controller). However, due to the constrained resources of the sensor nodes or IoT devices, the number of messages exchanged is minimized, and the messages are optimized.

1) Controller-to-SDVS message type: Initiated by the SD-IoT controller, and may or may not require a response. The messages for installation of forwarding and configuring instructions on the SDVS need no responses from the SDVS. This category includes messages as SetForwardingInstruction, SetConfiguringInstruction, ModifyConfiguration packets. However, if the controller demands for a SDVS's attributes or status, it needs the SDVS's responses.

a) SetForwardingInstruction/SetConfiguringInstruction: Enable the controller to install a forwarding/configuring instruction on a SDVS's forwarding/configuring table, and to respond to a SDVS's requests for a forwarding/configuring instruction respectively.

b) ModifyConfiguration: Modify a configuration instruction.

c) RequestFwdStats/RequestConfigStats: Get statistics of a forwarding or configuring instruction respectively.

d) ResponseFwdStats/ResponseConfigStats: Sent from a SDVS to the controller whenever the SDVS receives a RequestFwdStats/RequestConfigStats message respectively. These messages include information about the statistics of an instruction table or an instruction in the table.

e) RequestFeatures/ResponseFeatures: Get a SDVS's information about its sensor service list, the services' status, or driver of the underlying node.

2) Asynchronus message type (SDVS-to-Controller): Sent from a SDVS to a controller without any request from the controller. It enables the SDVS to ask for instruction on handling incoming packets as well as to update the controller on changes of its underlying sensor nodes regarding their active/deactive status, or completion of a required task.

a) Report packet: Report the status and behaviour of a SDVS. Particularly, the controller will be updated by changes as follows.

- Update the controller on its features (ReportFeatures).
- Inform the controller about the removal of a configuration instruction from a configuring table (ReportConfigurationRemove).
- Notify the controller about a sensor node's battery level (ReportLowBatt).
- Notify the controller that a sensor is at its maximum level of handling requests, so it is unavailable in the considering list of the controller (ReportFullTask).
- Inform the controller about the completion of a required service by a SDVS (ReportCompletion).

b) Response message type: Send required services back to a required destination.

c) Request message type: Request an instruction for its operation. Particularly, if a SDVS cannot find an instruction for handling an incoming packet, it sends a Request packet to the controller using its global knowledge of underlying network elements to respond to the request.

3) Symmetric message type (Controller/SDVS-to-SDVS/Controller): initiated by the controller or a SDVS, and sent periodically without solicitation from the other.

Hello message: This message is for a SDVS to notify its existence and for the controller to inform the SDVS that it does not receive an update for the period.

C. Forwarding table specifications

Similar to the OpenFlow table design, the forwarding table contains instruction entries as rows of the table. This table comprises three main components, matching window, action window, and statistic window (as presented in Fig. 3). The matching window is matched against an incoming packet. If a match is found, a corresponding action in the action window is executed, then associated statistics are updated for the matching packet. Otherwise, the packet is forwarded to the controller. The controller figures out how to process this packet.

1) Matching window: Provide information for extracting needed values from an arriving packet header. The extracted values are matched against the specified values in this window to find a match for the incoming packet. Thus, the window comprises four parameters.

a) ID: Indicate an ID of a matching window of an instruction. It is used when an incoming packet needs to be matched with many matching fields since each forwarding entry allows matching of a field in a packet header. It enables multiple header fields of an incoming packet to be considered, while it does not require more memory for storing multiple matching windows for an instruction entry.

b) Matching Field: Indicate which part of packet header is compared to the specified value in the matching window, which means that not all packet header fields are necessarily matched against a forwarding entry.

c) Operator: Indicate a comparison method between the header fields and matching fields. Operator values can be equal (=), different from (!=), higher than (>), higher than or equal to (>=), less than (<), less than or equal to (<=).

d) Value: Is compared to the extracted header field.

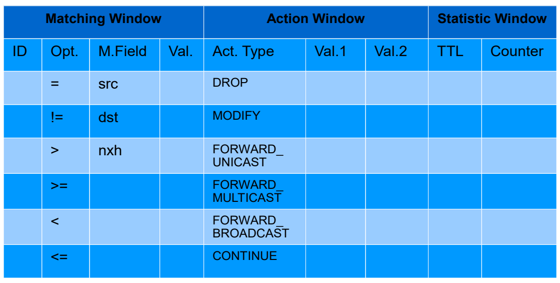
2) Action window: Indicate a corresponding action for an instruction entry. There are three parts which are Action Type, value 1, and value 2.

a) Action Type: Indicate a type of action. Possible action types are as FORWARD_UNICAST, FORWARD_MULTICAST, FORWARD_BROADCAST, DROP, MODIFY, or CONTINUE.

b) Value 1: Different action types result in the different meaning of the Value 1. For example, the MODIFY action type requires a new value and the modified value. As for the CONTINUE action, the forwarding instruction ID needs to be specified, so the incoming packet needs to be matched against the instruction entry with the same ID. For the FORWARD_UNICAST, FOWARD_MULTICAST, and FOWARD_BROADCAST action type, it demands the unicast, multicast, and broadcast address respectively.

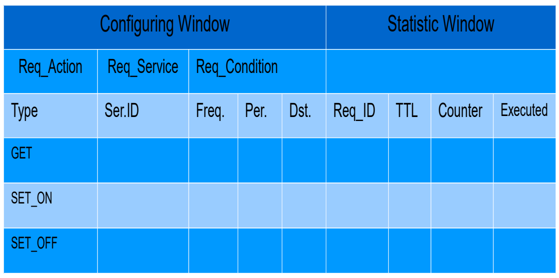
c) Value 2: A replacement for the old value.

3) Statistic window: with a focus on efficiently programming of underlying sensors/ IoT devices, statistics



Matching Window				Action Window			Statistic Window	
ID	Opt.	M.Field	Val.	Act. Type	Val.1	Val.2	TTL	Counter
	=	src		DROP				
	!=	dst		MODIFY				
	>	nxh		FORWARD_ UNICAST				
	>=			FORWARD_ MULTICAST				
	<			FORWARD_ BROADCAST				
	<=			CONTINUE				

Fig. 3: Forwarding table structure



Configuring Window					Statistic Window			
Req_Action	Req_Service	Req_Condition						
Type	Ser.ID	Freq.	Per.	Dst.	Req_ID	TTL	Counter	Executed
GET								
SET_ON								
SET_OFF								

Fig. 4: Configuring table structure

would enable the update of sensor and network status. When a match is found, statistics related to the matched instruction is updated. The statistics record information regarding Time To Live (TTL) and Counter.

a) TTL: Is a time to live of a forwarding instruction entry. It is decreased when the instruction table is updated. Its value depends on the required amount of time of an application request. It is gradually reduced to zero and is deleted when reaching zero.

b) Counter: Count the number of packets matched against a forwarding entry.

D. Configuring table specifications

The configuring table provides a SDVS with instructions about configuration for its underlying sensors/IoT devices. Its structure is composed of two main windows, Configuring, and Statistic (depicted in Fig. 4).

1) Configuring window: The configuring window includes three components: required services, required condition, and required action.

a) Required service: Indicate the required sensor service.

b) Required conditions: Indicate the conditions related to the required service.

- *Frequency:* Specify how often the required sensor service is achieved.
- *Period:* Is an executing period of an instruction
- *Destination Address:* Specify the destination of results returning by a SDVS. If there is no specified value, the destination is the controller.

c) Required action: Indicate an action type applied to the required service under the specified conditions.

2) Statistic window: Show the number of configuring instructions, and their operating time associated with an application request. Its elements include request ID, TTL, Counter, and Executed status.

a) Request ID: Indicate that the configuring instructions belonging to an application request. When the last configuring instruction of a ReqID is executed, the SDVS send an acknowledgment to the controller about its completion of a required task.

b) TTL: is the existing time of a configuring instruction, and is defined by application requirements. When it reaches zero, the related instruction is removed.

c) Counter: Show the current number of requests for a sensor service, and the busy level of a SDVS.

d) Executed: Specify if an instruction is executed.

V. IMPLEMENTATION OF THE SD-IOT MODEL

A. Implementation setup

The implementation is developed from our preliminary implementation [14] and the open source SDN-WISE platform [15] into a comprehensive design and implementation. The SD-IoT model is a software platform written in Java and built in Netbeans 8.2. It is connected to a database built in MySQL. The three main elements are the SD-IoT controller, the S-MANAGE protocol, and the SDVS.

Three software modules, control, southbound interface, and virtual representation are responsible for the SD-IoT controller, the S-MANAGE, and the SDVS respectively. The control module includes classes responsible for analyzing application requests, orchestrating SDVS resources, generating instructions relating to the requests, networking and communicating with the SDVS. The Southbound interface module is composed of classes for S-MANAGE messages, forwarding table, configuring table. The virtual representation module contains classes for defining the core of a SDVS and its software-defined function.

We also establish a database in MySQL to store and update information regarding the SDVS in the network, such as its sensor services, status, location, and attributes. The database provides data for an operation of resource orchestrator in the controller.

B. Implementation prototype

We build a network where the controller has one-hop connections to five SDVS. With the prototype, the aim is to demonstrate the feasibility of the proposed S-MANGE protocol in programming the SDVS. The implementation prototype is shown in the Fig. 5.

C. Implementation results

Users send requests to the controller via a GUI interface. The request parameters indicate the required services (SID), required timing for the services (Freq., Per.), the location of the required services (LOCID), a destination for the returned results (Dst.), identification of the request (ReqID), and the action for the required services (Act_Type). With the request, the controller orchestrates its resources to select appropriate SDVSs to handle the demand.

Request = Act_Type, SIDs, Freq, Per, Dst., LOCID, ReqID

The following figures illustrate a forwarding table, a configuring table, and a sensor service list of a SDVS in the network of 5 SDVSs in two cases: i) before it is configured by the controller, and ii) after it is configured according to a request for its sensor services. The SDVS is SDVS02 with the address 202.

Fig.6, Fig.8, and Fig.10 demonstrate results for the first case. They present the status of the SDVS02 before it is programmed by the controller, such as its forwarding instructions (Fig.6) and configuring instructions (Fig.8), and sensor services status (Fig.10).

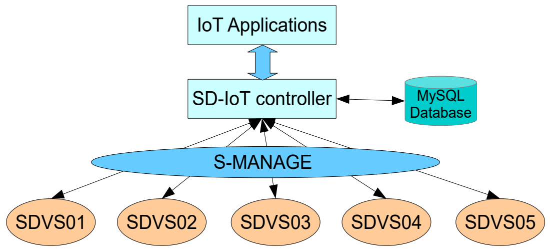
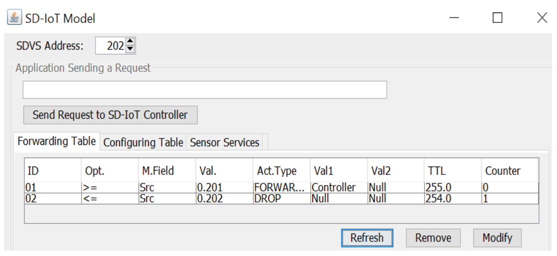


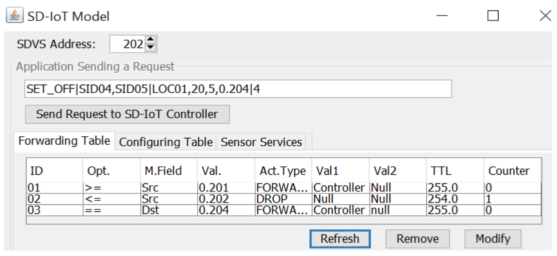
Fig. 5: Implementation prototype



SDVS Address: 202

ID	Opt.	M.Field	Val.	Act.Type	Val1	Val2	TTL	Counter
01	>=	Src	0.201	FORWAR...	Controller	Null	255.0	0
02	<=	Src	0.202	DROP	Null	Null	254.0	1

Fig. 6: Forwarding table status before configuration

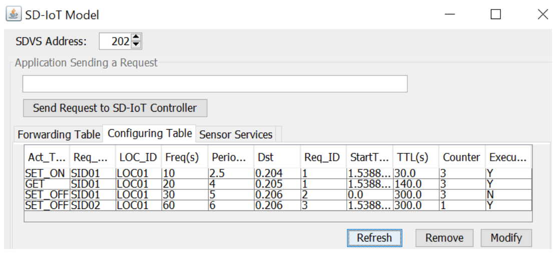


SDVS Address: 202

SET_OFF|SID04,SID05|LOC01,20,5,0.204|4

ID	Opt.	M.Field	Val.	Act.Type	Val1	Val2	TTL	Counter
01	>=	Src	0.201	FORWA...	Controller	Null	255.0	0
02	<=	Src	0.202	DROP	Null	Null	254.0	1
03	==	Dst	0.204	FORWA...	Controller	null	255.0	0

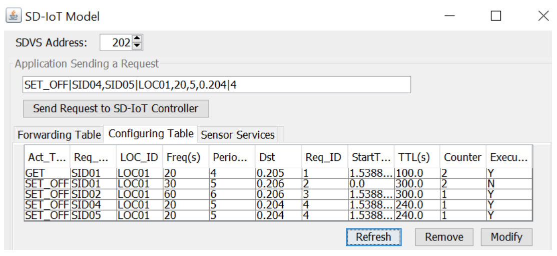
Fig. 7: Forwarding table status after configuration



SDVS Address: 202

Act_T...	Req_...	LOC_ID	Freq(s)	Perio...	Dst	Req_ID	StartT...	TTL(s)	Counter	Execu...
SET_ON	SID01	LOC01	10	2.5	0.204	1	1.5388...	30.0	3	Y
GET	SID01	LOC01	20	4	0.205	1	1.5388...	140.0	3	Y
SET_OFF	SID01	LOC01	30	5	0.206	2	0.0	300.0	3	N
SET_OFF	SID02	LOC01	60	6	0.206	3	1.5388...	300.0	1	Y

Fig. 8: Configuring table status before configuration

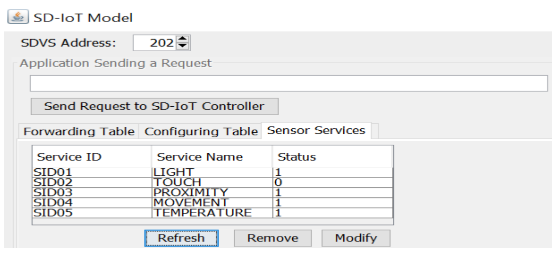


SDVS Address: 202

SET_OFF|SID04,SID05|LOC01,20,5,0.204|4

Act_T...	Req_...	LOC_ID	Freq(s)	Perio...	Dst	Req_ID	StartT...	TTL(s)	Counter	Execu...
GET	SID01	LOC01	20	4	0.205	1	1.5388...	100.0	2	Y
SET_OFF	SID01	LOC01	30	5	0.206	2	0.0	300.0	2	N
SET_OFF	SID02	LOC01	60	6	0.206	3	1.5388...	300.0	1	Y
SET_OFF	SID04	LOC01	20	5	0.204	4	1.5388...	240.0	1	Y
SET_OFF	SID05	LOC01	20	5	0.204	4	1.5388...	240.0	1	Y

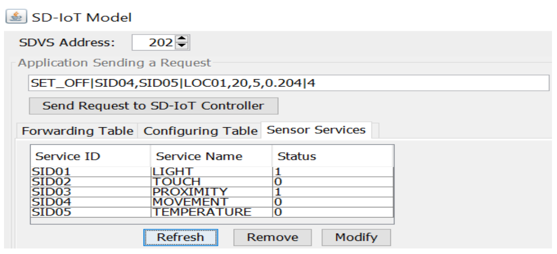
Fig. 9: Configuring table status after configuration



SDVS Address: 202

Service ID	Service Name	Status
SID01	LIGHT	1
SID02	TOUCH	0
SID03	PROXIMITY	1
SID04	MOVEMENT	1
SID05	TEMPERATURE	1

Fig. 10: Sensor service status before configuration



SDVS Address: 202

SET_OFF|SID04,SID05|LOC01,20,5,0.204|4

Service ID	Service Name	Status
SID01	LIGHT	1
SID02	TOUCH	0
SID03	PROXIMITY	1
SID04	MOVEMENT	0
SID05	TEMPERATURE	0

Fig. 11: Sensor service status after configuration

Fig.7, Fig.9, and Fig.11 reveal results for the second case. When there is a request for deactivating sensor services (SID04 and SID05), the SDVS02 is configured by the controller according to the request. The controller deploys associated instructions to the SDVS's forwarding table (Fig.7), configuring table (Fig.9). Thus, its sensor services are configured, and their status changed accordingly from 1(activate) to 0 (deactivate) (Fig.11).

The presented results only aim to demonstrate the main features of the S-MANGAGE southbound protocol from the SD-IoT controller.

VI. Conclusion

In this paper, we propose a design and implementation of the S-MANAGE protocol to address the challenges of configuring and programming IoT devices in provisioning IoT applications on demand. The S-MANAGE is designed based on OpenFlow and OF-CONFIG to IoT devices in both forwarding behaviours and their functionalities. Details of the design are provided. We also propose a SDVS as an interface enabling the controller to control and manage physical IoT devices via the S-MANAGE protocol. The implementation performances demonstrate the feasibility of the proposed protocol. The proposal enables further research and development on interoperability and orchestration of heterogeneous WSN/IoT devices for the provision of diverse IoT application on demand.

References

[1] Li, S., L.D. Xu, and S. Zhao, 5G Internet of Things: A survey. Journal of Industrial Information Integration, 2018. 10: p. 1-9.

[2] Bera, S., S. Misra, and A.V. Vasilakos, Software-Defined Networking for Internet of Things: A Survey. IEEE Internet of Things Journal, 2017. 4(6): p. 1994-2008.

[3] Luo, T., H.-P. Tan, and T.Q.S. Quek, Sensor OpenFlow: enabling software-defined wireless sensor networks. Communications Letters, IEEE, 2012. 16(11): p. 1896-1899.

[4] Razzaque, M.A., et al., Middleware for Internet of Things: A Survey. IEEE Internet of Things Journal, 2016. 3(1): p. 70-95.

[5] Kobo, H.I., A.M. Abu-Mahfouz, and G.P. Hancke, A Survey on Software-Defined Wireless Sensor Networks: Challenges and Design Requirements. IEEE Access, 2017. 5: p. 1872-1899.

[6] Costanzo, S., et al. Software defined wireless networks: unbridling sdns. in European Workshop on Software Defined Networking (EWSDN). 2012. IEEE.

[7] Galluccio, L., et al. SDN-WISE: design, prototyping and experimentation of a stateful SDN solution for WIreless SEnsor networks. in 2015 IEEE Conference on Computer Communications (INFOCOM). 2015.

[8] 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.

[9] Mahmud, A. and R. Rahmani. Exploitation of OpenFlow in wireless sensor networks. in 2011 International Conference on Computer Science and Network Technology (ICCSNT). 2011.

[10] Anadiotis, A.-C.G., et al., SD-WISE: a software-defined wireless sensor network. arXiv preprint arXiv:1710.09147, 2017.

[11] Bera, S., et al., Soft-WSN: Software-Defined WSN Management System for IoT Applications. IEEE Systems Journal, 2016. PP(99): p. 1-8.

[12] Foundation, O.N., OpenFlow Switch Specification, in Version 1.0.0 (Wire Protocol 0x01). 2009.

[13] Foundation, O.N., OpenFlow Configuration and Management Protocol OF-CONFIG 1.0. 2011.

[14] Nguyen, T.M.C., D.B. Hoang, and T.D. Dang. A software-defined model for IoT clusters: Enabling applications on demand. in 2018 International Conference on Information Networking (ICOIN). 2018.

[15] Milardo, S. The stateful Software Defined Networking solution for the Internet of Things. 2017; Available from: https://github.com/sdnwiselab/sdn-wise-java.