S-MANAGE Protocol for Provisioning IoT Applications on Demand

Thi Minh Chau Nguyen
University of Technology Sydney, Faculty of Engineering & IT

Doan B. Hoang
University of Technology Sydney, Faculty of Engineering & IT

Abstract: Internet of Things (IoT)-based services have started making an impact in various domains, such as agriculture, smart farming, smart cities, personal health, and critical infrastructures. Sensor/IoT devices have become one of the indispensable elements in these IoT systems and services. However, their development is restricted by the rigidity of the current network infrastructure, which accommodates heterogeneous physical devices. Software-Defined Networking-Network Functions Virtualization (SDN-NFV) has emerged as a service-enabling solution, supporting network and network function programmability. Provisioning IoT applications on demand is a natural application of programmability. However, these technologies cannot be directly deployed in the sensing/monitoring domain 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. This paper proposes an S-MANAGE protocol that preserves the SDN-NFV paradigm but provides a practical solution in controlling and managing IoT resources for provisioning IoT applications on demand. S-MANAGE is proposed as a new southbound protocol between the software-defined IoT controller and its IoT elements. The paper presents the design of S-MANAGE and demonstrates its use in provisioning IoT services dynamically.

Keywords: Provisioning services on demand, Software-defined IoT model, Programming services, Network functions virtualization, Software-defined virtual sensor (SDVS)

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, actuating capabilities, and resources. The number of IoT devices in the world is predicted to be about 6.58 billion by 2020 (Perera *et al.*, 2014). Deployment of an IoT application can become challenging owing to large geographical coverage of the application, limitation of resources of IoT devices, heterogeneity of the environment, and a huge number of these devices (Li, Xu & Zhao, 2018). 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 (Li, Xu & Zhao, 2018). Therefore, many efforts have been put into programming IoT devices to meet IoT application demands (Javed *et al.*, 2018).

Among solutions to the programmability of Wireless Sensor Networking/Internet of Things (WSN/IoT) systems, many proposals have taken advantage of the Software-Defined Networking (SDN) paradigm (Bera, Misra & Vasilakos, 2017). The SDN provides solutions to programmability, agility, flexibility and end-to-end connectivity challenges, which are associated with management of real-time traffic flows and dynamic traffic patterns (Deva Priya & Silas, 2019). The SDN approach addresses many existing problems concerning network management and provisioning resources required by network services. It can change the functionality of physical networks as well as devices in real time to meet requirements of IoT applications (Sood, Yu & Xiang, 2016). The SDN principle separates the network control plane from the data plane of networking devices and allows the provision of on-demand services through a programmable and logically centralised controller. Autonomous management of network devices is enabled under SDN. The SDN architecture comprises three main planes, which are application plane, control plane, and data plane. The control plane centrally controls and manages the behaviour of the whole network 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, challenges remain when applying the SDN paradigm to the constrained WSN/IoT (Luo, Tan & Quek, 2012). As a fundamental element of the underlying resources that provide necessary data for IoT applications, an IoT system must of necessity 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. While many attempts have been made to address IoT platform architectures and to provision IoT applications on demand, challenging issues remain: they include scalable and dynamic resource discovery and composition; context-awareness; integration of intelligence; interoperability; reliability; security and privacy; and system-wide scalability (Razzaque *et al.*, 2016). In this paper, we propose the S-MANAGE protocol as an enabler of the software-defined Internet of Things (SD-IoT) model, suggested in our previous work (Nguyen, Hoang & Dang, 2017).

The SD-IoT model adopts SDN and Network Functions Virtualization (NFV) principles and deploys these technologies to IoT devices to provide IoT applications on demand. However, it is not feasible to completely and directly apply the SDN technique to the resource-limited WSN/IoT environment, owing to its constraints (Kobo, Abu-Mahfouz & Hancke, 2017). In the model, the NFV technique is deployed to realise software-defined virtual sensors as a representation of the underlying IoT resources. This technology is thus applied readily to the WSN/IoT environment for creating a virtual representation of IoT devices that are utilised by multiple IoT applications simultaneously. This virtual representation offers a solution to enrich the features of limited IoT devices. By applying both SDN and NFV principles in the proposed SD-IoT model, diverse underlying sensor nodes can be programmed in accordance with IoT application requests.

In the SD-IoT model, the S-MANAGE protocol has been proposed as a communication bridge between the SD-IoT controller and the software-defined virtual sensor (SDVS) in each cluster. The controller sets up and configures the SDVS by using S-MANAGE, which is designed to deal with the constraints of IoT systems. It should be emphasised that the SDN OpenFlow protocol was designed to handle SDN network (routing) devices and it is not suitable for resource-constrained IoT devices, whose mission is different from that of SDN routers and switches. Furthermore, a separate protocol such as OF-CONFIG is often required to configure the network devices, and this introduces complexity to already constrained IoT devices. This paper investigates the design and the implementation of S-MANAGE. S-MANAGE is designed both to configure SDVSs and to control the behaviour of the underlying networked IoT resources. Major contributions of this paper are as follows:

1. It proposes a new southbound S-MANAGE protocol for programming the behaviour and configurational management of software-defined virtual sensors and their associated physical devices.
2. It proposes a programmability approach to provisioning IoT applications on demand.
3. It describes an implementation of the proposed protocol in the context of a software-defined Internet of Things system and provides implementation results.

The remainder of the paper is organised as follows. Section II reviews related work. 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 an implementation scenario and evaluates its performance. Section VI concludes the paper.

Related Work

The most challenging task in applying the SDN paradigm to the WSNs/IoT environment is the design of the communication interface between the SDN-based controller and underlying IoT devices. However, existing proposals mainly suggest modification to the well-known OpenFlow SBI without an actual implementation: for example, Sensor OpenFlow (Luo, Tan & Quek, 2012), and SDWN (Costanzo *et al.*, 2012). The feasibility of application of SDN to WSNs has been demonstrated in the SDN-WISE (Galluccio *et al.*, 2015) proposal. The SDN-WISE SBI has been designed in accord with the OpenFlow protocol, thus enabling programmability of a WSN sink node's forwarding behaviour. 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 node's forwarding behaviour without concern for programming a sensor node's functionality. The details of these proposals are discussed in our previous work (Nguyen, Hoang & Chaczko, 2016).

Another work (Mahmud & Rahmani, 2011) deploys the OpenFlow technology in a WSN to enable a share of IoT resources for larger scale networks. They propose flow-sensors that communicate with an access point via the OpenFlow protocol. Their implementation results demonstrate how OpenFlow can be of benefit in controlling and monitoring sensor traffic flow, but there is no effort to make the OpenFlow protocol suitable for constrained sensor nodes.

Device and network management has been considered by the soft-WSN (Bera *et al.*, 2016) proposal. The proposed architecture is in accordance with the SDN paradigm and provides a network and device management approach for provisioning IoT application-aware services. However, the focus is on the design of the controller and the sensor node architecture. The controller is designed with two management policies, topology and device management. The communication between the two entities is based on traditional protocols, IEEE 802.15.4 and IEEE 802.11. There is no effort to solve the complexity of deployment of flow tables to the sensor nodes.

Software-Defined IoT Model

To reap the benefits of the SDN paradigm, the SD-IoT model is also structured in three layers – application, control, and data – as depicted in Figure 1.

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

The control layer accommodates the SD-IoT controller and its database. 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 control the underlying IoT resources. At the same time, it provides the underlying resources with an interface for updating their status and attributes, as well as their sensor services. With the knowledge of both requirements for and capabilities of the IoT resources, it can provide sensor services for IoT applications on demand.

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 IoT devices. The virtual data layer enables the controller to manage and control the underlying IoT resources in the physical data layer.

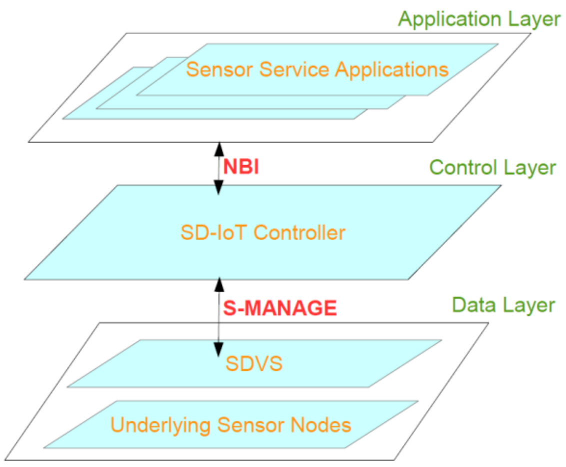


Figure 1. SD-IoT architecture

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.

Specifically, the SD-IoT controller makes it possible for the application layer to specify its demands via an 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 the devices.

S-MANAGE protocol

The S-MANAGE protocol is proposed as an SBI between the SD-IoT controller and the virtual data layer. The protocol allows the controller to communicate with software-defined virtual sensors (SDVSs) in this layer. Moreover, via S-MANAGE, the controller can collect statistics regarding the physical IoT devices. 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 behaviour of SDVSs.

Software-defined virtual sensor (SDVS)

Software-defined virtual sensors (SDVSs) are defined as representatives of physical/software sensor nodes or IoT devices located in the physical layer. It is configured with core features and attributes of a physical sensor node, and with a software-defined function (SDF). The core modules enable the SDVS to behave as the represented physical IoT device; the modules can interact with the represented devices via device-specific protocols. In addition, the SDF allows the controller to enhance the SDVS with processing, computing or forwarding functions. In particular, the forwarding and configuring functions are implemented in the SDVS as SDFs. The SDVS is connected to its underlying IoT device and acts on behalf of the represented device. In addition, it is installed with S-MANAGE protocol features, so it can communicate with the controller. In summary, an SDVS can be considered as a software layer that enriches its represented physical sensor/IoT device, allowing the SD-IoT controller to configure the device and program its behaviour. The design and implementation of SDVSs is the topic of another paper.

S-MANAGE Protocol

In traditional IP networks, routers are used to relay packets (or datagrams) according to the 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, including transport layer ID, network layer IP address, VLAN ID, MAC layer ID, and router port IP address. As a consequence, a flow in the OpenFlow protocol requires some 12 matching parameters to be identified. 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; direct deployment of OpenFlow in WSN/IoT networks is not appropriate. Furthermore, the OpenFlow SDN network still requires OF-CONFIG or other protocols for device configuration. What we need is a streamlined protocol in WSN/IoT networks that can handle both configuration of the IoT devices and simple types 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 SBI, the controller can both manage and configure the SDVS in this layer.

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

The protocol is proposed according to the spirit of two protocols, OpenFlow (ONF, 2012a) and OF-CONFIG (ONF, 2012b). OpenFlow focuses on flow rules for setting, modification, and deletion, or adding rules for controlling forwarding behaviour of OpenFlow switches. Meanwhile, 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 IoT devices to be based on two proposed instruction tables, called forwarding and configuring tables. The forwarding table instructs an SDVS on how to handle an arriving packet, while the configuring table guides the SDVS to configure its represented underlying nodes.

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

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

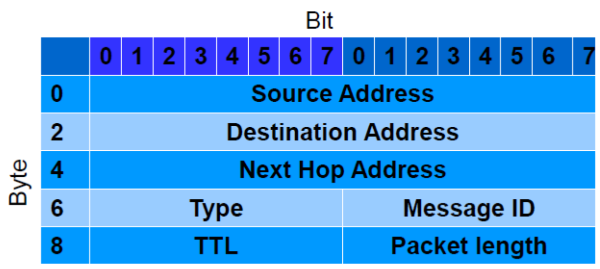
S-MANAGE defines communication methods between the controller and an SDVS. It specifies exchanged message types between the two entities, the message format, the structure of instruction tables, and how the SDVS is programmed and should operate based on these tables' instructions. Details of the protocol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

S-MANAGE packet header

The S-MANAGE packet is comprised of a header and a payload. All S-MANAGE messages begin with an S-MANAGE header, as depicted in Figure 2. The header size is 10 bytes. It includes the following parts:

- 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 (1 byte) is an identifier of the packet type.



Bit

Byte	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
0	Source Address															
2	Destination Address															
4	Next Hop Address															
6	Type								Message ID							
8	TTL								Packet length							

Figure 2. S-MANAGE packet header

Message types

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

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 minimised, and the messages are optimised.

Controller-to-SDVS message type

This message type is 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 such as SetForwardingInstruction, SetConfiguringInstruction, and ModifyConfiguration packets. However, if the controller demands an SDVS's attributes or status, it needs the SDVS's responses.

a) SetForwardingInstruction/SetConfiguringInstruction: Enable the controller to install a forwarding/configuring instruction on an SDVS's forwarding/configuring table, and to respond to an 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 an 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 an SDVS's information about its sensor service list, the services' status, or the driver of the underlying IoT device.

Asynchronous message type (SDVS-to-Controller)

The message type is sent from an SDVS to the 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 in its underlying sensor nodes regarding their active/idle status or completion of a required task.

a) Report packet: Report the status and behaviour of an SDVS. Particularly, the controller will be updated by changes as follows.
   a. Update the controller on its features (ReportFeatures).
   b. Inform the controller about the removal of a configuration instruction from a configuring table (ReportConfigurationRemove).
   c. Notify the controller about a sensor node's battery level (ReportLowBatt).
   d. Notify the controller that a sensor is at its maximum level of handling requests, so it is unavailable to be assigned further tasks by the controller (ReportFullTask).
   e. Inform the controller about the completion of a required service by an SDVS (ReportCompletion).

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

c) Request message type: An SDVS requests for 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, which uses its global knowledge of underlying network elements to respond to the request.

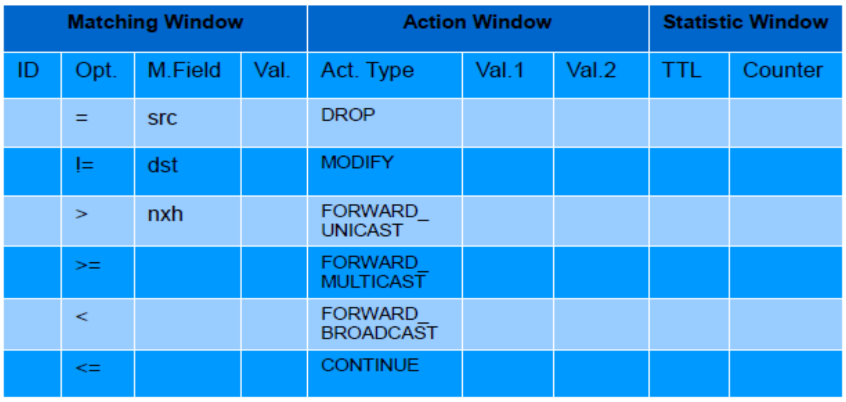
Symmetric message type (Controller/SDVS-to-SDVS/Controller)

This message type is initiated by the controller or an SDVS and sent periodically without solicitation from the other.

Hello message: This message is for an SDVS to notify its existence and for the controller to inform the SDVS that it has not received an update for the current period.

Forwarding table specifications

The forwarding table contains instruction entries as rows of the table. This table is composed of three main elements: matching window, action window, and statistic window (as presented in Figure 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 the packet.



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				

Figure 3. Forwarding table structure

Matching window

It provides information for extracting needed values from an arriving packet header. The extracted values are matched against the specified values in the window to find a match for the incoming packet. The window is comprised of four parameters:

a) ID: Indicates 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: Indicates 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: Indicates a comparison method between the matching header field and the matching window Value. 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.

Action window

The window indicates a corresponding action for an instruction entry. The action window is composed of three parts: Action Type, value 1, and value 2. The value 1 and value 2 parts do not have a specific name, since they may represent values of different matching fields according to the action-type value.

a) Action Type: Indicates a type of action. Possible action types are FORWARD UNICAST, FORWARD MULTICAST, FORWARD BROADCAST, DROP, MODIFY, or CONTINUE.
b) Value 1: Different action types result in different meanings of 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. The FORWARD UNICAST, FOWARD MULTICAST, and FOWARD BROADCAST action types demand the unicast, multicast, and broadcast address, respectively.
c) Value 2: A replacement for the old value.

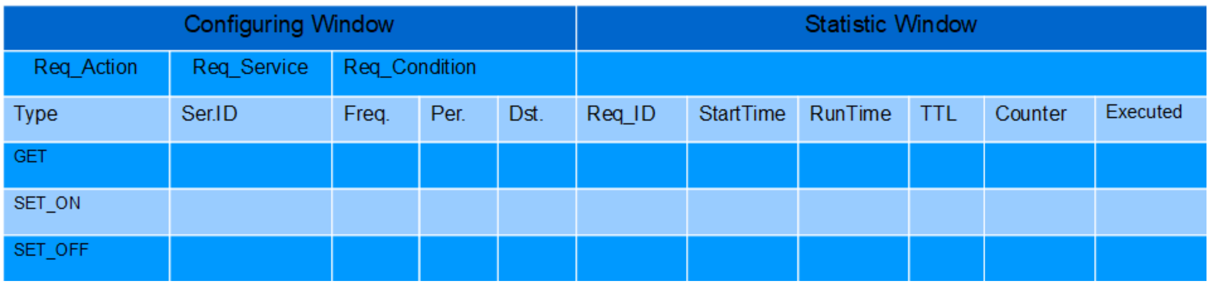
Statistic window

With a focus on efficiently programming of underlying IoT devices, their forwarding statistics would be necessary for an update of the network status. When a match is found, statistics related to the matched instruction are updated. The statistics are about 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: Counts the number of packets matched against a forwarding entry.

Configuring table specifications

The configuring table provides an SDVS with instructions about configuration for its underlying IoT devices. Its structure is composed of two main windows: configuring and statistics (as presented in the Figure 4).



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

Figure 4. Configuring table structure

Configuring window

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

a) Required service: Indicates the required sensor service.
b) Required conditions: Indicates the conditions related to the required service.

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

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

Statistic window

The window shows the number of configuring instructions, and their operating time associated with an IoT application request. It includes information associated with an instruction, namely request ID, TTL, Counter, Operation time, and Executed status.

a) Request ID: Indicates which application request is associated with the configuring instruction. When the last configuring instruction of a ReqID is executed, the SDVS sends an acknowledgment to the controller about its completion of the 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: Shows the current number of requests for a sensor service and is used for updating a state of an SDVS. The state indicates a busy level of the SDVS. The higher the state number, the busier is the SDVS. The state is computed according to the total number of tasks that an SDVS performs and is updated in accordance with the counter statistics.
d) Operation time: Shows timing data related to an execution of an IoT application request. It provides information about the starting time and running time of an executed request. The information is essential for the orchestration function of the controller.
e) Executed: Specifies if an instruction has been executed or not. The executed status marked with "Y" means executed and with "N" means not executed.

Implementation of S-MANAGE in Provisioning IoT Sensor Services for IoT Applications

An implementation scenario

Our aim is provisioning IoT applications on demand by using the S-MANAGE protocol in the context of the SD-IoT model. Any request for IoT services is dynamically processed by the SD-IoT model. The system can orchestrate its underlying resources to handle multiple simultaneous sensor service demands (as shown in the Figure 5). According to its knowledge of the capability of the underlying resources, the system can i) obtain the availability of the resources and their current service-provisioning tasks; ii) provide appropriate responses to an application request, such as meet the request fully, or suggest an alternative that satisfies the request partially, or unable to provide the services because of insufficient resources; iii) handle simultaneous application requests and deal with conflicts among these requests; and iv) collect results corresponding to each application request.

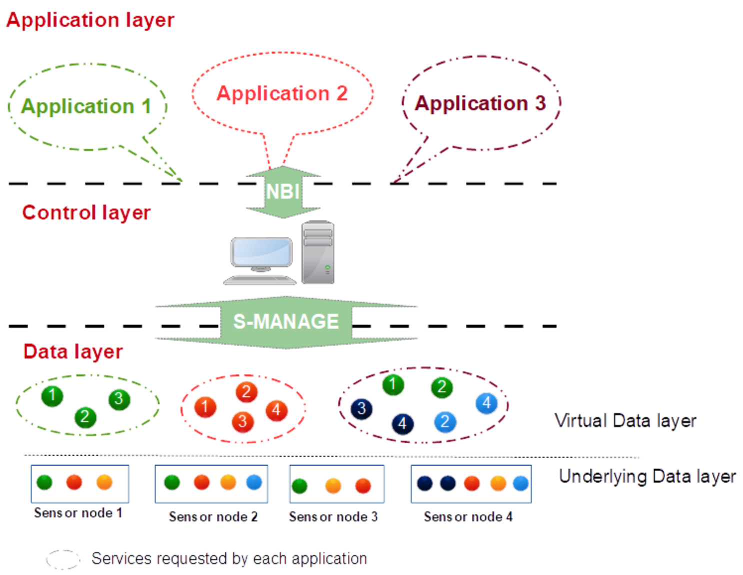


Figure 5. Implementation scenario

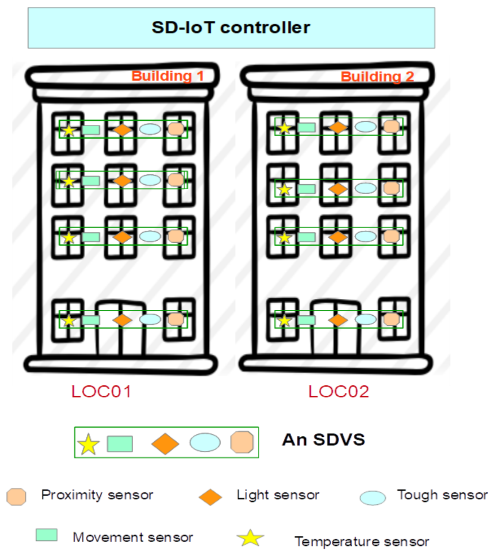


Figure 6. Implementation use case

For the sake of demonstration of a practical realization of the proposed protocol, we deploy the SD-IoT model that controls and manages two clusters of sensor nodes. The two resources are in two different locations. They can be orchestrated to provide sensor services for one or multiple IoT applications on demand. For the case study, the two clusters represent two buildings within a campus. Each building has four floors. Many types of sensors may be used on different floors, such as movement, temperature, proximity, touch, and light sensors (as presented in Figure 6).

A GUI interface is designed to enable users to indicate their sensor service types of interest and also to make specific demands for the required services. For example, they can indicate sensor services of interest, how long and how often they want to obtain the services. Moreover, they can choose the destination for the required services. An IoT application request is comprised of a set of these requirements.

Implementation setup

The implementation is developed from our preliminary implementation (Nguyen, Hoang & Dang, 2018). We also make use of Java dependencies support from the open source SDN-WISE platform (Milardo, 2017). To realise the operation of S-MANAGE, we build the SD-IoT model in which S-MANAGE provides a communication approach between the SD-IoT controller and its IoT elements. 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. We have implemented three main software components of the proposed SD-IoT model: the SD-IoT controller, the S-MANAGE protocol, and the SDVS.

- The control module includes classes responsible for analysing application requests, orchestrating SDVS resources, generating instructions relating to the requests, networking and communicating with the SDVS.
- The Southbound interface module comprises classes for the construction of S-MANAGE messages, forwarding tables, and configuring tables of SDVSs.
- The virtual representation module is composed of classes for initiating instances of an SDVS.

We build a network where the controller communicates with its SDVSs. We establish a database in MySQL to store and update information regarding the SDVSs in the network, such as their sensor services, status, location, and attributes. The statistics from the forwarding and configuring tables are used to update the attributes, the status of the SDVSs and their underlying IoT devices. The database provides essential information for an operation of the controller's core modules.

Implementation results

Implementation results demonstrate the expected features of the proposed S-MANAGE protocol in provisioning IoT applications on demand. S-MANAGE makes it possible for the controller to instruct IoT devices to achieve required services as well as forward results to required destinations. In addition, the protocol enables the controller to collect statistical information from the underlying IoT resources. Therefore, the controller can achieve the following results.

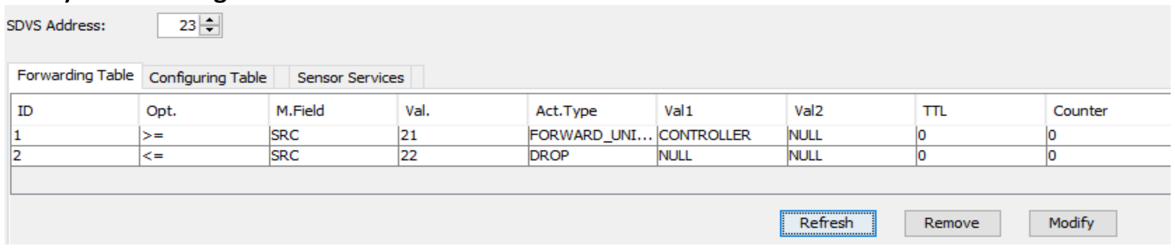
i) Programming its IoT resources via S-MANAGE according to an application request (Figure 7 and Figure 8).

ii) Responding dynamically to an application request about the service provisioning capability of the system according to its residual resources (as shown in Figure 9).

iii) Handling simultaneous application requests and conflicts over these requests (as demonstrated in Figure 10).

iv) Obtaining and displaying the status of multiple on-going application requests (as presented in Figure 11).

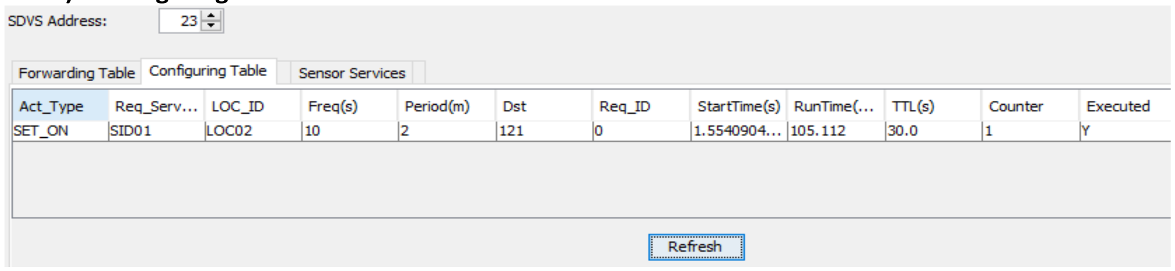
a) Forwarding table



SDVS Address: 23

ID	Opt.	M.Field	Val.	Act.Type	Val1	Val2	TTL	Counter
1	>=	SRC	21	FORWARD_UNI...	CONTROLLER	NULL	0	0
2	<=	SRC	22	DROP	NULL	NULL	0	0

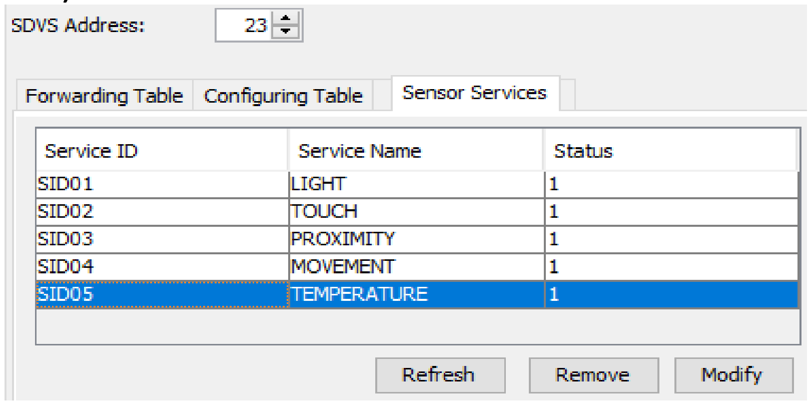
b) Configuring table



SDVS Address: 23

Act_Type	Req_Serv...	LOC_ID	Freq(s)	Period(m)	Dst	Req_ID	StartTime(s)	RunTime(...	TTL(s)	Counter	Executed
SET_ON	SID01	LOC02	10	2	121	0	1.5540904...	105.112	30.0	1	Y

c) Sensor services status

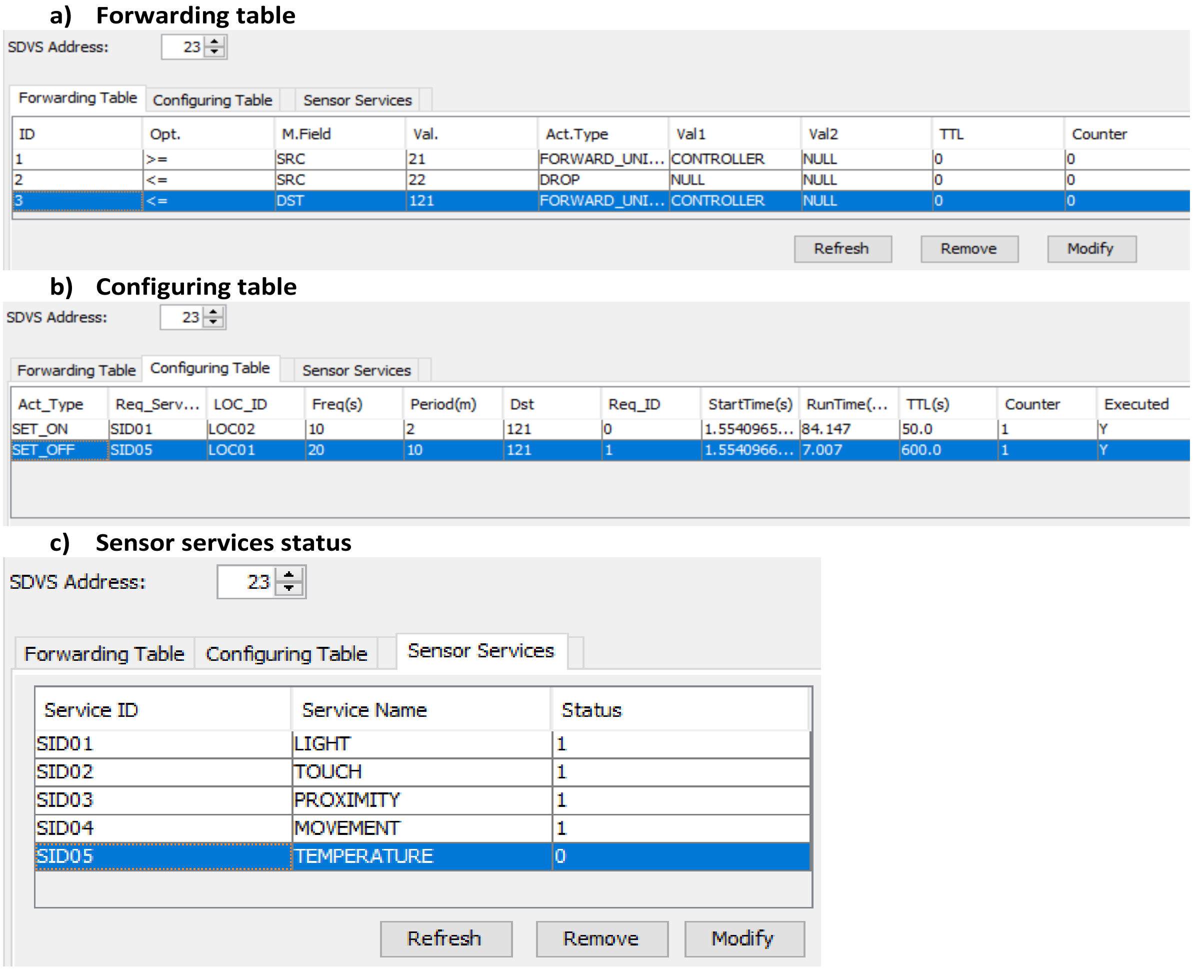


SDVS Address: 23

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

Figure 7. Status of the SDVS before its configuration

The programmable function of S-MANAGE is demonstrated in Figure 7 and Figure 8. The two figures illustrate the status of an SDVS (SDVS03) before and after, respectively, it is programmed by the controller. In each figure, the status of the SDVS is presented, its forwarding instructions in (a), configuring instructions in (b), and sensor services status in (c). Differences between Figure 7 and Figure 8 are: i) both the forwarding and configuring tables of the SDVS03 are installed with one new instruction entry; and ii) changes in the status of the required service belonging to the SDVS. Via the installed configuring instruction, the SDVS can achieve the required services. Deploying the forwarding instruction, it knows how to forward results to the required destination. The result for the request is to change the status of the sensor service SID05 from 1 (ON) (as shown in Figure 7-c) to 0 (OFF) (as shown in Figure 8-c).


a) Forwarding table

SDVS Address: 23

Forwarding Table | Configuring Table | Sensor Services

ID	Opt.	M.Field	Val.	Act.Type	Val1	Val2	TTL	Counter
1	>=	SRC	21	FORWARD_UNI…	CONTROLLER	NULL	0	0
2	<=	SRC	22	DROP	NULL	NULL	0	0
3	<=	DST	121	FORWARD_UNI…	CONTROLLER	NULL	0	0

Refresh Remove Modify

b) Configuring table

SDVS Address: 23

Forwarding Table | Configuring Table | Sensor Services

Act_Type	Req_Serv…	LOC_ID	Freq(s)	Period(m)	Dst	Req_ID	StartTime(s)	RunTime(…	TTL(s)	Counter	Executed
SET_ON	SID01	LOC02	10	2	121	0	1.5540965…	84.147	50.0	1	Y
SET_OFF	SID05	LOC01	20	10	121	1	1.5540966…	7.007	600.0	1	Y

c) Sensor services status

SDVS Address: 23

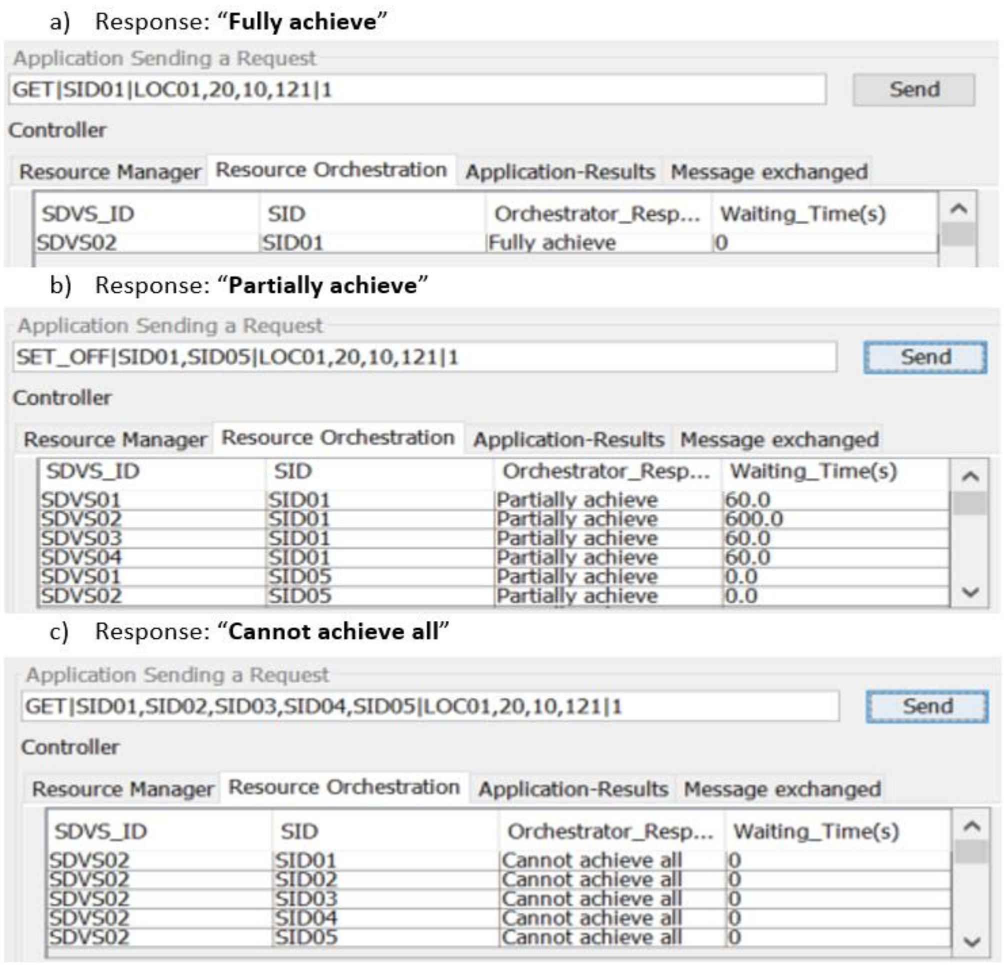
Forwarding Table | Configuring Table | Sensor Services

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

Refresh Remove Modify

Figure 8. Status of the SDVS after its configuration

Moreover, thanks to the S-MANAGE protocol, the controller can muster the available IoT resources and orchestrate them to satisfy all the services whenever demanded. The S-MANAGE messages allow the controller to collect essential information about the updated status of the underlying IoT resources. If a request can be partially provisioned, the controller will also inform the application. Depending on the reply from the application, the controller performs its tasks based on the status table containing status of all SDVSs. The controller can program appropriate SDVSs to handle an incoming request according to its status (availability and capability). As shown in Figure 9, the controller provides appropriate responses to the application request in the case i) it can fully achieve all the required services (see Figure 9-a); ii) it partially achieves the required services and provides waiting time for obtaining the remaining required services (see Figure 9-b); or unable to provide the services because of insufficient resources (see Figure 9-c).


a) Response: “**Fully achieve**”

Application Sending a Request

GET|SID01|LOC01,20,10,121|1

Send

Controller

Resource Manager | Resource Orchestration | Application-Results | Message exchanged

SDVS_ID	SID	Orchestrator_Resp...	Waiting_Time(s)
SDVS02	SID01	Fully achieve	0

b) Response: “**Partially achieve**”

Application Sending a Request

SET_OFF|SID01,SID05|LOC01,20,10,121|1

Send

Controller

Resource Manager | Resource Orchestration | Application-Results | Message exchanged

SDVS_ID	SID	Orchestrator_Resp...	Waiting_Time(s)
SDVS01	SID01	Partially achieve	60.0
SDVS02	SID01	Partially achieve	600.0
SDVS03	SID01	Partially achieve	60.0
SDVS04	SID01	Partially achieve	60.0
SDVS01	SID05	Partially achieve	0.0
SDVS02	SID05	Partially achieve	0.0

c) Response: “**Cannot achieve all**”

Application Sending a Request

GET|SID01,SID02,SID03,SID04,SID05|LOC01,20,10,121|1

Send

Controller

Resource Manager | Resource Orchestration | Application-Results | Message exchanged

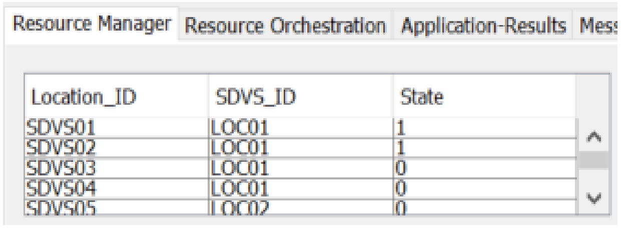
SDVS_ID	SID	Orchestrator_Resp...	Waiting_Time(s)
SDVS02	SID01	Cannot achieve all	0
SDVS02	SID02	Cannot achieve all	0
SDVS02	SID03	Cannot achieve all	0
SDVS02	SID04	Cannot achieve all	0
SDVS02	SID05	Cannot achieve all	0

Figure 9. Dynamic response from the controller’s resource orchestration to an application request

In addition, the system can handle multiple simultaneous application requests and resolve conflicts among these requests. As presented in Figure 10, in the control panel of the controller, the Resource Manager tab shows SDVSs’ locations and their state. The Application-Results tab presents the current application requests and the status of the SD-IoT model’s IoT application provision. Figure 10 illustrates three different states of the SDVS’s functionality and corresponding tasks. In state 1, SDVS01 and SDVS02 are providing services SID01 and SID05 for the two application requests 1 and 2, respectively. Meanwhile, in state 2, SDVS02 receives another application request number 3 for the service SID05. The request cannot be processed immediately owing to the conflict between the two requests 2 and 3 for the same service. The request number 2 requires data from the sensor service, but the request number 3 requires deactivating the sensor service Therefore, SDVS02 delays the request number 3 until it completes the request number 1. In state 3, when the request number 1 is done, SDVS02 achieves the required service for the request number 3.

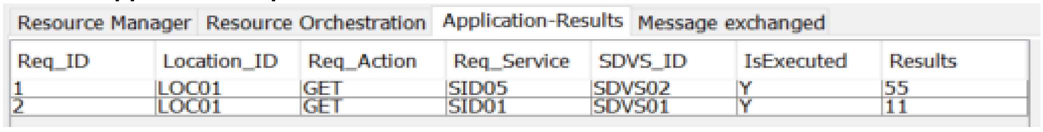
State 1: application requests and current-task status of involved SDVSs.

Current status of each SDVS



Location_ID	SDVS_ID	State
SDVS01	LOC01	1
SDVS02	LOC01	1
SDVS03	LOC01	0
SDVS04	LOC01	0
SDVS05	LOC02	0

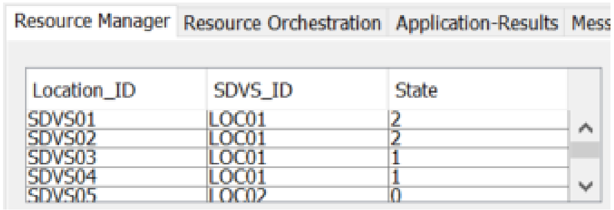
Current application requests and related executed status



Req_ID	Location_ID	Req_Action	Req_Service	SDVS_ID	IsExecuted	Results
1	LOC01	GET	SID05	SDVS02	Y	55
2	LOC01	GET	SID01	SDVS01	Y	11

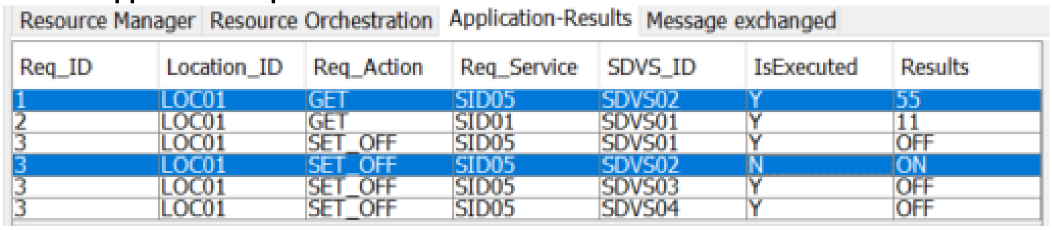
State 2: when there is an incoming request to turn off the required service SID05, all SDVSs in LOC01 have to be reconfigured. However, the SDVS01 is currently providing SID05 for another application: SDVS01 cannot start processing the incoming request for SID05.

Current status of each SDVS



Location_ID	SDVS_ID	State
SDVS01	LOC01	2
SDVS02	LOC01	2
SDVS03	LOC01	1
SDVS04	LOC01	1
SDVS05	LOC02	0

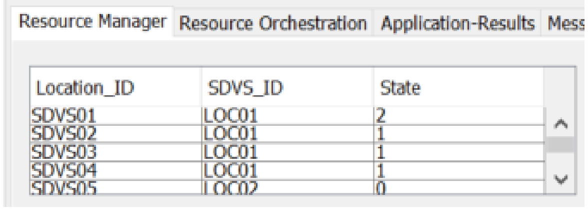
Current application requests and related executed status



Req_ID	Location_ID	Req_Action	Req_Service	SDVS_ID	IsExecuted	Results
1	LOC01	GET	SID05	SDVS02	Y	55
2	LOC01	GET	SID01	SDVS01	Y	11
3	LOC01	SET_OFF	SID05	SDVS01	Y	OFF
3	LOC01	SET_OFF	SID05	SDVS02	N	ON
3	LOC01	SET_OFF	SID05	SDVS03	Y	OFF
3	LOC01	SET_OFF	SID05	SDVS04	Y	OFF

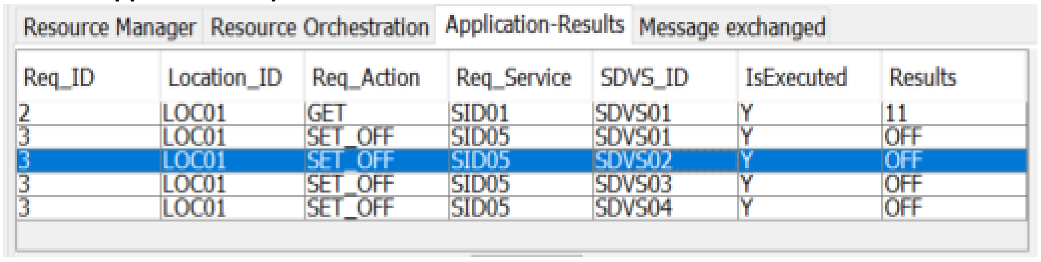
State 3: After releasing the task for request number 1, the SDVS02 processes the request number 3.

Current status of each SDVS



Location_ID	SDVS_ID	State
SDVS01	LOC01	2
SDVS02	LOC01	1
SDVS03	LOC01	1
SDVS04	LOC01	1
SDVS05	LOC02	0

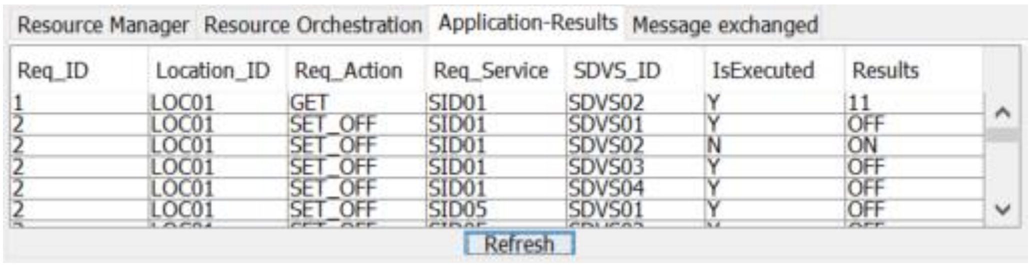
Current application requests and related executed status



Req_ID	Location_ID	Req_Action	Req_Service	SDVS_ID	IsExecuted	Results
2	LOC01	GET	SID01	SDVS01	Y	11
3	LOC01	SET_OFF	SID05	SDVS01	Y	OFF
3	LOC01	SET_OFF	SID05	SDVS02	Y	OFF
3	LOC01	SET_OFF	SID05	SDVS03	Y	OFF
3	LOC01	SET_OFF	SID05	SDVS04	Y	OFF

Figure 10. Handling multiple application requests and solving conflicts among them

Figure 11 shows the status of all application requests and associated results. The Application-Results tab presents information about all application requests (represented by the Req_ID) and their execution status (see IsExecuted column: Y means Executed and N means Not-Executed). Moreover, the tab also displays required parameters regarding service type, location, related action, and associated results (see the Results column).



Req_ID	Location_ID	Req_Action	Req_Service	SDVS_ID	IsExecuted	Results
1	LOC01	GET	SID01	SDVS02	Y	11
2	LOC01	SET_OFF	SID01	SDVS01	Y	OFF
2	LOC01	SET_OFF	SID01	SDVS02	N	ON
2	LOC01	SET_OFF	SID01	SDVS03	Y	OFF
2	LOC01	SET_OFF	SID01	SDVS04	Y	OFF
2	LOC01	SET_OFF	SID05	SDVS01	Y	OFF

Figure 11. Status of on-going application requests and corresponding results

Conclusion

In this paper, we propose a design and implementation of the S-MANAGE protocol in order to address challenges of configuring and programming an IoT network and devices in provisioning IoT applications on demand. S-MANAGE is designed to configure functionalities of resource-constrained IoT devices through their virtual representations (SDVSs) and program their forwarding behaviours. Details of the design are provided. The implementation performance demonstrates the feasibility of the proposed protocol and its application. The proposal also enables further research and development on interoperability and orchestration of heterogeneous WSN/IoT devices for the provision of diverse IoT applications on demand.

References

Bera, S., Misra, S., Roy, S. K., & Obaidat, M. S. (2016). Soft-WSN: Software-Defined WSN Management System for IoT Applications. *IEEE Systems Journal, 12*(3), 2074-2081. doi:10.1109/JSYST.2016.2615761

Bera, S., Misra, S., & Vasilakos, A. V. (2017). Software-Defined Networking for Internet of Things: A Survey. *IEEE Internet of Things Journal, 4*(6), 1994-2008. doi:10.1109/JIOT.2017.2746186

Costanzo, S., Galluccio, L., Morabito, G., & Palazzo, S. (2012). Software Defined Wireless Networks: Unbridling SDNs. *2012 European Workshop on Software Defined Networking (EWSDN)*, October. doi:10.1109/EWSDN.2012.12

Deva Priya, I., & Silas, S. (2019). A Survey on Research Challenges and Applications in Empowering the SDN-Based Internet of Things. In: Peter, J., Alavi, A., & Javadi, B. (eds), *Advances in Big Data and Cloud Computing*, Advances in Intelligent Systems and Computing, vol. 750, Singapore: Springer. doi:10.1007/978-981-13-1882-5_39

Galluccio, L., Milardo, S., Morabito, G., & Palazzo, S. (2015). SDN-WISE: design, prototyping and experimentation of a stateful SDN solution for WIreless SEnsor networks. *2015 IEEE Conference on Computer Communications (INFOCOM)*, April-May. doi: 10.1109/INFOCOM.2015.7218418

Javed, F., Afzal, M. K., Sharif, M., & Kim, B. (2018). Internet of Things (IoT) Operating Systems Support, Networking Technologies, Applications, and Challenges: A Comparative Review. *IEEE Communications Surveys & Tutorials, 20*(3), 2062-2100. doi:10.1109/COMST.2018.2817685

Kobo, H. I., Abu-Mahfouz, A. M., & Hancke, G. P. (2017). A Survey on Software-Defined Wireless Sensor Networks: Challenges and Design Requirements. *IEEE Access, 5*, 1872-1899. doi:10.1109/ACCESS.2017.2666200

Li, S., Xu, L. D., & Zhao, S. (2018). 5G Internet of Things: A survey. *Journal of Industrial Information Integration, 10*, 1-9. doi:10.1016/j.jii.2018.01.005

Luo, T., Tan, H.-P., & Quek, T. Q. S. (2012). Sensor OpenFlow: Enabling Software-Defined Wireless Sensor Networks. *IEEE Communications Letters, 16*(11), 1896-1899. doi: 10.1109/LCOMM.2012.092812.121712

Mahmud, A., & Rahmani, R. (2011). Exploitation of OpenFlow in wireless sensor networks. *Proceedings of 2011 International Conference on Computer Science and Network Technology*, December. doi: 10.1109/ICCSNT.2011.6182029

Milardo, S. (2017). The stateful Software Defined Networking solution for the Internet of Things. Retrieved from https://github.com/sdnwiselab/sdn-wise-java

Nguyen, T. M. C., Hoang, D. B., & Chaczko, Z. (2016). Can SDN Technology Be Transported to Software-Defined WSN/IoT? *2016 IEEE International Conference on Internet of Things (iThings)* and *IEEE Green Computing and Communications (GreenCom)* and *IEEE Cyber, Physical and Social Computing (CPSCom)* and *IEEE Smart Data (SmartData)*, December. doi: 10.1109/iThings-GreenCom-CPSCom-SmartData.2016.63

Nguyen, T. M. C., Hoang, D. B., & Dang, T. D. (2017). Toward a programmable software-defined IoT architecture for sensor service provision on demand. *2017 27th International Telecommunication Networks and Applications Conference (ITNAC)*, Melbourne, November. doi: 10.1109/ATNAC.2017.8215419

Nguyen, T. M. C., Hoang, D. B., & Dang, T. D. (2018). A software-defined model for IoT clusters: Enabling applications on demand. *2018 International Conference on Information Networking (ICOIN)*, January. doi: 10.1109/ICOIN.2018.8343223

ONF [Open Networking Foundation]. (2012a). OpenFlow Switch Specification, Version 1.3.0 (Wire Protocol 0x04), ONF TS-006, 25 June. Retrieved from https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf

ONF [Open Networking Foundation]. (2012b). OpenFlow Management and Configuration Protocol (OF-Config 1.1), Version 1.1, ONF TS-005, 25 June. Retrieved from https://www.opennetworking.org/wp-content/uploads/2013/02/of-config-1.1.pdf

Perera, C., Liu, C. H., Jayawardena, S., & Min, C. (2014). A Survey on Internet of Things From Industrial Market Perspective. *IEEE Access, 2*, 1660-1679. doi:10.1109/ACCESS.2015.2389854

Razzaque, M. A., Milojevic-Jevric, M., Palade, A., & Clarke, S. (2016). Middleware for Internet of Things: A Survey. *IEEE Internet of Things Journal, 3*(1), 70-95. doi:10.1109/JIOT.2015.2498900

Sood, K., Yu, S., & Xiang, Y. (2016). Software-Defined Wireless Networking Opportunities and Challenges for Internet-of-Things: A Review. *IEEE Internet of Things Journal, 3*(4), 453-463. doi:10.1109/JIOT.2015.2480421