

A Large-Scale Software-Defined Internet of Things Platform for Provisioning IoT Services on Demand

A thesis submitted in fulfillment of the requirements for
the degree of Doctor of Philosophy
in the Faculty of Engineering and Information Technology
at the University of Technology Sydney

by

Thi Minh Chau Nguyen

Supervised by

Professor Doan B. Hoang

2020

Certificate of Original Authorship

I, Thi Minh Chau Nguyen, declare that this thesis is submitted in fulfillment of the requirements for the award of the degree of Doctor of Philosophy in the Faculty of Engineering and Information Technology at the University of Technology Sydney.

This thesis is wholly my own work unless otherwise reference or acknowledged. In addition, I certify that all information sources and literature used are indicated in this thesis.

This document has not been submitted for qualifications at any other academic institution.

This research is supported by the Australian Government Research Training Program.

Signature: Production Note:
 Signature removed
 prior to publication. Thi Minh Chau Nguyen

Date: 15/04/2020

Dedication

To my parents, aunty, and siblings

To my primary supervisor

Thank for your great support

Acknowledgment

During my doctoral candidature, I have received a myriad of lessons, support, and encouragement. First of all, I would like to express my sincere gratitude to my primary supervisor, Professor Doan B. Hoang, for his precious lessons and patient guidance. He has taught me significantly valuable lessons that cannot be found from books or any document. I have learned not only research and academic skills but also problem-solving skills from him. Starting from zero research background, I have gradually become an independent researcher. His understanding, enthusiasm, advice, and supervision have pushed me farther than I thought I could go. Without his critical comments, insightful feedback, encouragement, and motivation, none of my work would have been possible and successful, and I would still be in the marsh of the academic career.

I wish to acknowledge the support of my co-supervisor, Dr. Zenon Chaczko. I am thankful for SEDE staff always helping me with admin procedures.

I would extend my thanks to the Mekong 1000 project and CTU-UTS (Can Tho University – University of Technology Sydney) scholarship that has offered me an opportunity to pursue the Ph.D. course.

I will never forget the encouragement and assistance from my colleagues and friends, including Tham Nguyen, Sara Farahmandian, Ngoc Le, Vinh Ha, Dat Dang, Thai Nguyen, Cheng te Wang, Nhut Huynh, Firas AI Dughman, Jan Szymanski, Ashish Nanda, and Deepak Puthal.

Finally, I wish to show my appreciation to my parents, aunty, and siblings, who are always by my side to encourage me and keep me going.

Without all of you, this research would not have been possible.

Sydney, 2020.

The Author's Publications

International Conference Publications and Proceedings:

1. **Nguyen, T.M.C.**, D.B. Hoang, and Z. Chaczko. "Can SDN Technology Be Transported to Software-Defined WSN/IoT?" in *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)*. 2016.
2. **Nguyen, T.M.C.**, D.B. Hoang, and T.D. Dang. "Toward a programmable software-defined IoT architecture for sensor service provision on demand." in *2017 27th International Telecommunication Networks and Applications Conference (ITNAC)*. 2017.
3. **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.
4. **Nguyen, T.M.C.**, and D.B. Hoang. "S-MANAGE Protocol for Software-Defined IoT." in *2018 28th International Telecommunication Networks and Applications Conference (ITNAC)*. 2018.
5. **Nguyen, T.M.C.**, and D.B. Hoang. "Software-Defined Virtual Sensors for Provisioning Services on Demand." in *2020 International Conference on Information Technology and Internet of Things (ITIOT)* (accepted for publication).

Journal papers:

6. **Nguyen, C.** and D. Hoang, "S-MANAGE Protocol for Provisioning IoT Applications on Demand." *Journal of Telecommunications and the Digital Economy*, 2019. 7(3): p. 37-57.
7. **Nguyen, C.** and D. Hoang, "Large-scale Software-Defined Internet of Things Platform for Provisioning IoT Services on Demand," submitted to *International Journal of Smart Sensor Technologies and Applications (IJSSTA)*. (Under 2nd round review) (2020)

Table of Contents

Certificate of Original Authorship	i
Dedication	ii
Acknowledgment	iii
The Author’s Publications	iv
List of Figures	ix
List of Tables	xii
List of Abbreviations and Acronyms	xii
Abstract	xv
Chapter 1 Introduction	1
1.1 Introduction	1
1.2 Brief Background	5
1.3 Research Questions	7
1.4 Research Aim and Objectives	10
1.5 Research Contributions and Significance	10
1.6 Research Methodology.....	12
1.7 Thesis Structure.....	13
Chapter 2 Background and Related Work	17
2.1 Introduction	17
2.2 IoT System	18
2.2.1 IoT Evolution Stages	18
2.2.2 Things, Services, and Resources in IoT	22
2.2.3 IoT System – Key Components.....	25
2.2.4 IoT Architecture	28
2.2.5 Requirements for an IoT System	32
2.3 IoT Deployment Models and Scenarios.....	34
2.3.1 IoT Deployment Models.....	34
2.3.2 Real IoT Scenarios	35
2.3.3 IoT Application Development - Challenges.....	36

2.4	Software-Defined Networking (SDN) Technique	38
2.4.1	SDN Architecture	39
2.4.2	SDN Paradigm's Implications to WSN/IoT	44
2.4.3	Challenges of Application of SDN to WSN/IoT	46
2.5	Network Function Virtualization (NFV) Technique.....	48
2.6	Solutions to a Programmable IoT Device	50
2.7	SDN-NFV-based Solutions to a Programmable IoT System.....	52
2.8	SDN-NFV-Based Solutions to a Large-Scale IoT System	54
2.9	Open-Sources for Developing LSSD-IoT Platform.....	55
2.10	Summary	56
Chapter 3 Large-Scale Software-Defined Internet of Things (LSSD-IoT)		
Model	57
3.1	Introduction	57
3.2	Why Large-Scale and Programmable Services on Demand?	58
3.3	LSSD-IoT Model	60
3.3.1	Software-Defined Cluster Layer.....	64
3.3.2	Software-Defined Device Layer.....	65
3.4	LSSD-IoT Features	67
3.5	Practical Realization of the Proposed LSSD-IoT Model	69
3.6	Thesis Roadmap	70
3.7	Summary	71
Chapter 4 Software-Defined Virtual Sensor (SDVS).....		
4.1	Introduction	72
4.2	Proposed SDVS.....	74
4.3	SDVS – Representation Types.....	77
4.4	SDVS Features	79
4.5	SDVS Architecture and Software Implementation	81
4.5.1	SDVS Architecture	81
4.5.2	Software Implementation	83
4.6	Use Case Scenario and Practical Implementation.....	91
4.7	Performance Evaluation	92
4.7.1	SDVS – Feasibility and Programmability	92

4.7.2 SDVS – Efficiency	95
4.8 Summary	97
Chapter 5 S-MANAGE Protocol.....	98
5.1 Introduction	98
5.2 S-MANAGE in Relation to SD-IoT Model	99
5.3 S-MANAGE Protocol	102
5.3.1 S-MANAGE Header	104
5.3.2 S-MANAGE Message types.....	105
5.3.3 Forwarding Table Specifications.....	108
5.3.4 Configuring Table Specifications.....	110
5.4 Software Implementation	111
5.5 Implementation and Performance Evaluation	115
5.5.1 Implementation Set up.....	115
5.5.2 Performance Evaluation	116
5.6 Summary	120
Chapter 6 Software-Defined Internet of Things (SD-IoT) Model.....	121
6.1 Introduction	121
6.2 SD-IoT Model	123
6.2.1 Software-Defined Virtual Sensor (SDVS)	124
6.2.2 S-MANAGE Protocol	124
6.3 SD-IoTD Controller	125
6.3.1 SD-IoTD Controller – Functional Components	125
6.3.2 SD-IoTD Controller – Operational Mechanism.....	126
6.4 SD-IoTD Controller - Software Implementation	129
6.5 SD-IoT Model – Software Implementation	131
6.5.1 Use Case Scenario	131
6.5.2 Implementation Scenario.....	132
6.5.3 Implementation Set up.....	133
6.6 Performance Evaluation	134
6.7 Summary	142
Chapter 7 Software-Defined Cluster Layer and LSSD-IoT Platform	143
7.1 Introduction	143

7.2	SD Cluster Layer.....	144
7.2.1	SD-IoTC Controller.....	145
7.2.2	SD-IoT Clusters and Communication with the SD-IoTC Controller.....	148
7.3	LSSD-IoT Platform – Procedure of the Provision of IoT Services on Demand.....	149
7.4	LSSD-IoT Platform – Use cases	151
7.5	LSSD-IoT Platform Implementation.....	157
7.5.1	Implemented Platform	157
7.5.3	Implementation Scenario.....	160
7.5.3	Implementation Set up.....	161
7.5.4	SD-IoTC Controller – Software Implementation.....	163
7.6	Performance Evaluation.....	170
7.6.1	Implementation Platform Capability	170
7.6.2	Platform Performance.....	180
7.7	Summary	185
Chapter 8	Conclusion and Future Work.....	187
8.1	Research Remarks.....	187
8.2	Future Work	191
	Appendices	193
	Bibliography	202

List of Figures

Figure 1.1 Research phases	12
Figure 1.2 Research Methodology	13
Figure 1.3 Thesis Organization	14
Figure 2.1 IoT Evolution.....	19
Figure 2.2 Relationship between IoT applications, services, and resources [36]	25
Figure 2.3 IoT building blocks and technologies	25
Figure 2.4 IoT service types.....	27
Figure 2.5 IoT architecture	29
Figure 2.6 IoT reference model [32]	31
Figure 2.7 Mapping of IoT requirements for enabling technologies	33
Figure 2.8 Real IoT Scenarios	35
Figure 2.9 SDN architecture.....	39
Figure 2.10 Three main components of an SDN switch and OpenFlow switch	41
Figure 2.11 NFV architecture	49
Figure 2.12 Mapping SDN components for NFV architecture.....	50
Figure 3.1 LSSD-IoT architecture	63
Figure 3.2 SD Cluster layer architecture	64
Figure 3.3 SD Device layer architecture.....	65
Figure 4.1 SDVS in relation to LSSD-IoT model.....	73
Figure 4.2 SDVS' s representation types	79
Figure 4.3 Sensor node architecture	82
Figure 4.4 SDVS architecture	83
Figure 4.5 SDVS in software	84
Figure 4.6 Class diagram of the SDVS	85
Figure 4.7 SDVS's java class	86
Figure 4.8 Basic parameters defining an SDVS and its represented entity.....	87
Figure 4.9 Abstract methods defining fundamental functions of an SDVS and its represented entity	90
Figure 4.10 Implementation prototype.....	92
Figure 4.11 SDVS's programmable features	93
Figure 4.12 Sensor tag's log file	94
Figure 4.13 Configuration status of SDVS21	95
Figure 4.14 SDVS's average response time for one request per multiple requests	96
Figure 5.1 S-MANAGE protocol in relation to the LSSD-IoT model.....	99
Figure 5.2 SD-IoT model	100

Figure 5.3 Sequence diagram for Forwarding Statistics achievement	104
Figure 5.4 S-MANAGE header	104
Figure 5.5 S-MANAGE message types	106
Figure 5.6 Forwarding table structure	108
Figure 5.7 Configuring table structure.....	110
Figure 5.8 Class diagram of the configuring table	112
Figure 5.9 Details of a configuring entry	113
Figure 5.10 Class diagram of the forwarding table	114
Figure 5.11 Class diagram of S-MANAGE packets	115
Figure 5.12 Implementation prototype.....	116
Figure 5.13 Forwarding table status before configuration	117
Figure 5.14 Forwarding table status after configuration.....	118
Figure 5.15 Configuring table status before configuration.....	118
Figure 5.16 Configuring table status after configuration	119
Figure 5.17 Sensor service status before configuration.....	119
Figure 5.18 Sensor service status after configuration	119
Figure 6.1 SD-IoT model in relation to the LSSD-IoT model.....	122
Figure 6.2 SD-IoT model	123
Figure 6.3 SD-IoTD controller structure.....	125
Figure 6.4 Class diagram of SD-IoTD controller	130
Figure 6.5 Use case scenario	132
Figure 6.6 Implementation prototype.....	133
Figure 6.7 Status of the SDVS before its configuration	135
Figure 6.8 Status of the SDVS after its configuration	136
Figure 6.9 Dynamic response from the controller’s resource orchestrator to an IoT request	137
Figure 6.10 Handling multiple application requests and solving conflicts among them	139
Figure 6.11 Status of ongoing application requests and corresponding results	140
Figure 6.12 SD-IoTD Controller – Processing time for one per multiple simultaneous requests ranging between 10-90.....	141
Figure 6.13 Number of exchanged control and data messages between SD-IoTD controller and SDVSs	142
Figure 7.1 SD cluster layer in relation to LSSD-IoT model.....	144
Figure 7.2 SD-IoTC controller architecture	145
Figure 7.3 Overall procedure of provisioning IoT services on demand via LSSD-IoT platform....	150
Figure 7.4 Workflow of the SD-IoTC Controller	151
Figure 7.5 Workflow of the SD-IoTD Controller	151
Figure 7.6 Monitoring air pollution use case	153
Figure 7.7 IoT service provision scenario number 1	153
Figure 7.8 Smart traffic control use case.....	154

Figure 7.9 IoT service provision scenario number 2	155
Figure 7.10 IoT service provision scenario number 3	156
Figure 7.11 Detailed implementation of LSSD-IoT architecture	159
Figure 7.12 LSSD-IoT Model - Implementation scenario.....	161
Figure 7.13 Detailed implementation of the LSSD-IoT platform.....	162
Figure 7.14 Class diagram of the SD-IoTC controller	164
Figure 7.15 RequestAnalyser class.....	165
Figure 7.16 SdiotModule class.....	167
Figure 7.17 Class diagram for REST-API	168
Figure 7.18 Table 1 - Connections to SD-IoT clusters.....	169
Figure 7.19 Table 2 – Available IoT services	169
Figure 7.20 Table 3 – SD-IoTC clusters’ capability	170
Figure 7.21 Level of programmability and orchestration of the LSSD-IoT platform.....	172
Figure 7.22 Control and management panel at the SD cluster and device level.....	173
Figure 7.23 Overview of the control and management at the cluster level	174
Figure 7.24 Overview of the control and management at the device level	174
Figure 7.25 Available SD-IoT resources.....	175
Figure 7.26 IoT Requests Status	176
Figure 7.27 IoT Application Announcements	176
Figure 7.28 SD-IoTC Controller – Resource Orchestration.....	177
Figure 7.29 SD-IoTD Controller – Resources Orchestration	178
Figure 7.30 SD-IoTD Controller – Device Configuration.....	180
Figure 7.31 Timing diagram.....	181
Figure 7.32 View of resource orchestration at the cluster level	182
Figure 7.33 Orchestration time with and without optimization	183
Figure 7.34 SD-IoTC Controller – Response Time	184
Figure 7.35 Orchestration and Response Time for optimization case with various input requests	185

List of Tables

Table 2.1 Requirements for an IoT system	32
Table 2.2 Differences between proposals to network architecture [45]	37
Table 7.1 Summary of IoT application requests and corresponding IoT services provided by IoT clusters (Ri represents a request from an IoT use case)	157

List of Abbreviations and Acronyms

AODV	Ad hoc On-Demand Distance Vector
API	Application Programming Interface
CASAGRAS	Coordination and Support Action for Global RFID-Related Activities and Standardization
CERP-IoT	Cluster of European Research Projects-IoT
CoT	Chain of Things
DEEC	Distributed Energy-Efficient Clustering
EM	Element Management
EPC	Electronic Product Code
FPGA	Field-Programmable Gate Array
ICN	Information-Centric Networking
IEEE	Institute of Electrical and Electronics Engineers
IERC	IoT European Research Cluster
IETF	Internet Engineering Task Force
IoT	Internet of Things
IP	Internet Protocol
IPSO	Internet Protocol for Smart Objects
IPv4	Internet Protocol version 4

IPv6	Internet Protocol version 6
ITS	Intelligent Transport Systems
ITU	International Telegraph Union
ITU-T	ITU Telecommunication Standardization Sector
ITU-TY.	ITU-T for machine learning
LEACH	Low-energy adaptive clustering hierarchy
LLDP	Link Layer Discovery Protocol
M2M	Machine to machine
MAC	Media Access Control
MANO	Management and Orchestration
MIT	Massachusetts Institute of Technology
NBI	Northbound Interface
NFC	Near-Field Communication
NFV	Network Function Virtualization
NFVI	NFV Infrastructure
NOS	Network Operating System
OS	Operating System
OvS	OpenvSwitch
REST/ RESTful	Representational State Transfer
RFID	Radio Frequency Identification
SBI	Southbound Interface
SD	Software-Defined
SD-IoT	Software-Defined Internet of Things
SD-IoTC	Software-Defined Internet of Things Cluster
SD-IoTD	Software-Defined Internet of Things Device
SDN	Software-Defined Networking
SDN-NFV	Software-Defined Networking and Network Function Virtualization
SDVS	Software-Defined Virtual Sensor
SDWSN	Software-Defined Wireless Sensor Network

SDWSN-RL	SDWSN-Reinforcement Learning
SIoT	Social Internet of Things
SOA	Service-Oriented Architecture
SOC	System on Chip
TCP	Transmission Control Protocol
TWh	terawatt-hours
UDP	User Datagram Protocol
uIP/uIPv6	Micro Internet Protocol/Micro Internet Protocol version 6
UWB	Ultra-Wide Band
VNF	Virtual Network Function
VNF	Virtual Network Function
VNFM	Virtual Network Function Manager
WSN	Wireless Sensor Network
WSN/IoT	Wireless Sensor Network or Internet of Things

Abstract

Internet of Things (IoT) has developed into an interconnected platform infrastructure for providing essential services ranging from personal health care, smart homes and cities to the manufacturing industry. Relying on such an infrastructure, a multitude of emerging IoT services will no doubt be developed for not only local regions but also multiple separated regions spreading over a wide geographical area. However, existing IoT systems are mostly rigid and cannot be easily adapted or programmed to accommodate new services. The challenge is also in orchestrating a large number of sensors/IoT devices, many with limited capability, into intelligent, useful, and on-demand services. Many efforts have been made to address the issue, but very little has been attempted to consider an overall solution to a programmable IoT ecosystem that includes IoT service provision components, IoT devices, and transporting infrastructure. Moreover, there is no framework/platform that allows an end-to-end control, management, and orchestration of IoT resources in accordance with IoT demands.

We apply the benefits of the two promising technologies including software-defined networking and network function virtualization in provisioning IoT services on demand over a wide region, and overcome challenges in applying the technologies to constrained IoT devices/systems. We propose a large-scale software-defined IoT (LSSD-IoT) model and develop the LSSD-IoT platform. The model provides two levels of management and orchestration at the cluster and device level. At the cluster level, we develop a software-defined Internet of Things Cluster (SD-IoTC) controller that is capable of controlling and managing both IoT clusters and network infrastructure that accommodates the IoT systems. At the device level, each IoT cluster under the control and management of the SD-IoTC controller needs to be programmable and manageable for provisioning IoT services on demand. For that purpose, we propose a software-defined Internet of Things (SD-IoT) model (local platform) with three novel components, including the IoT device-constrained controller, the S-MANAGE protocol, and the software-defined virtual sensor.

The novelty of this research lies in the novel approach to programmable and re-usable devices in the provision of IoT services on demand over a wide area. It enables i) IoT service providers to control end-to-end quality of services of IoT services provision over

a large-scale IoT environment; ii) owners of IoT systems to be able to gain benefits from sharing their IoT resources; iii) IoT application developers to develop innovative and comprehensive IoT applications on demand with more options regarding QoS, security, mobility, or billing.

Chapter 1 Introduction

1.1 Introduction

The “Internet” has changed our world and brought with it many technical, economic, and social benefits by connecting people. It is expected that the “Internet of Things” will create enormous value by interconnecting people and everyday things. In fact, IoT has already enabled many emerging applications and services critical to our life in various domains, from personal healthcare to smart cities, critical infrastructures, and supply chain logistics. It is projected that there will be a double growth in the combined market of the IoT from \$235 billion spent in 2017 to \$520 billion in 2021 [1]. However, this enormous potentiality is limited by existing IoT systems/platforms, which are mainly closed ecosystems that are vertically developed and deployed in their own IoT infrastructure and have incompatible standards, formats, semantics, and proprietary protocol and interfaces [2]. As a consequence, a number of major issues have been identified with the current generation of IoT systems/platforms:

The astronomical number of devices and their connectivity-service infrastructure. As projected, there would be about 41.6 billion IoT devices in 2025 [3]. The challenge here is how to manage the complexity of the interconnecting infrastructure of these devices and effectively serve both local communities and global communities distributed over a very large geographical area.

The massive number of IoT services and their provisioning framework. IoT devices are capable of interacting with their environment, performing their designated functions as well as collaborating with other IoT devices. Many services have already emerged to take advantage of these capabilities. The challenge is to automate the provisioning of these services whenever they are needed.

The vast amount of resources and their resource sharing. Collectively, through interconnectivity, IoT systems/platforms present a massive amount of resources and services to be shared among them. The challenge is in the developing of algorithms and supporting infrastructure for efficient use of the resources through sharing and reusing.

Most current IoT systems are not designed for sharing their resources among multiple applications. To support a specific application, a dedicated IoT infrastructure is designed with application-specific requirements for communication and computing capability, deploying many IoT devices or sensor nodes, and gateways. A number of application-specific IoT architectures were studied in [4]. It was found that to adapt an existing IoT application to a new one would demand a very complex effort in re-designing or re-configuring the underlying infrastructure as well as the network architecture. It becomes impractical to manually reconfigure individual devices that are equipped with fixed control logic. It also results in inefficient utilization of resources since it is difficult to dynamically optimize data acquisition, transmission, and processing. In addition, the deployment is more complex, and the maintenance cost is much higher due to a vast number of devices being deployed without sharing resources among applications. Furthermore, it takes much time to design and deploy an IoT infrastructure for developing a new IoT application.

The explosive development of IoT systems exposes a serious concern over the efficiency of the utilization of IoT resources and facilities for supporting the development of IoT applications. The increase in the number of IoT devices and collected data volume results in high demand for data centers that support IoT systems in processing the collected data in provisioning better IoT applications. A huge amount of money is being spent on the deployment of data centers, while only about 10% of the generated data are used by IoT applications. Besides being costly, data centers cause environmental impacts. In 2017, they consumed about 200 terawatt-hours (TWh), which is higher than the energy consumption of some nations [5]. To enable efficient utilization of IoT resources, the resources have to be shared among services, and the services have to be shared among themselves to serve their own cluster locally as well as other interconnected clusters regionally, nationally, and even globally. However, a large-scale IoT network cannot be

efficiently controlled and managed to provide resources on demand over multi-domain networks (e.g., Software-Defined Networking (SDN), cloud, edge, and IoT domains.)

Many approaches have been proposed to address the challenge of the provision of IoT services based on the integration of various IoT systems over a large-scale IoT infrastructure. The problem is examined on different aspects such as interoperability of IoT platforms, programmability of IoT devices and IoT networks, emerging web of things, platform as a service, or sensing as a service. Other efforts have been put into the adoption of the cloud and IoT for sharing IoT services in developing new IoT services and applications [6]. As defined by NIST [7], the cloud computing model has five remarkable characteristics, including measured service, rapid expansion or elasticity, resource pooling, broad network access, and on-demand self-service. These features can benefit the provision of IoT services on demand by reusing IoT resources connected to the cloud-based IoT infrastructure. On-demand IoT services bring benefits to things owners, computing, or network providers, the technical perspective, users as well as the entire IoT community [8].

A key enabler in the provision of IoT services on demand is the capability of deployment, management, and reconfiguration of an end-to-end process of achievement of IoT services in accordance with IoT requests. The process requires not only the *programmability at IoT devices level* but also the *routing of data and IoT services* at the transport network level for delivery and management of services to IoT applications or to the end-users which may be located in a Cloud or a remote location. Another requirement is the need for management of big data across the network as well as data gathered at the collection points, management of a huge number of IoT devices over a large-scale IoT infrastructure, and scalability management of local IoT systems. This becomes challenging, owing to the lack of a commonly agreed mechanism for the implementation of real collaborative IoT applications [9]. An additional challenge is the absence of efficient schemes for provisioning IoT sensing information on demand in a decentralized manner with consideration of dynamic changes of the physical world [10].

Software-Defined Networking and Network Function Virtualization (SDN-NFV) technologies have emerged as promising candidates for the programmability of the core network with central control and management of networking devices. The SDN paradigm

enables the SDN controller to have a global view of a network and to be able to program networking devices centrally. However, it is impractical to directly interact with millions of sensor nodes or IoT devices over the wide area ranging from wired to wireless domain. Therefore, the hierarchical architecture of network management can be seen as a potential solution to reduce the communication messages between the control plane and the data plane. A local IoT system can be centrally managed and controlled by a software controller. With the overall proposed software-defined architecture, the SDN controller can centrally manage and control many SDN domains (multiple IoT clusters), and the Software-Defined IoT local controller can manage its own individual IoT cluster when necessary resources are required by end-to-end IoT applications. This architecture thus allows resource management and orchestration of not only a local IoT system but also a large-scale network, including SDN and IoT environments.

This research addresses these challenges by proposing a large-scale software-defined Internet of Thing (LSSD-IoT) model for provisioning IoT services on demand. The model will include a hierarchical architecture with three layers: the top layer is the application layer which houses end-users' applications and interacts with the cluster layer to request end-to-end services, the cluster layer is for providing services that are distributed over multiple clusters covering a wide geographical area, and the device layer is for housing programmable platforms that serve local clusters. To the best of our knowledge, there is a lack of a mechanism for auto-orchestration of IoT services on demand over a large-scale IoT system using SDN and NFV techniques.

In order to realize such an IoT model, we address a number of challenging issues. We develop a new protocol for adapting and enhancing a Software-Defined Networking paradigm to IoT networks due to the different nature of network devices and IoT devices. We enrich the capability of IoT devices with function and interface virtualization for orchestrating and programming services. We develop algorithms and mechanisms for service orchestration and resource sharing.

The contribution of this research is the proposed large-scale software-defined IoT framework, the architecture for connectivity, the mechanisms for provisioning services on demand, and protocols and algorithms for orchestrating and sharing resources and services among IT clusters for deploying IoT applications on demand. The architecture

allows the local SD-IoT systems to be easily integrated into a large-scale IoT infrastructure but still preserve the simplicity and economy of sensors or IoT devices. More importantly, the benefits of software-defined centralized control, virtualization, programmability, and autonomous management and configuration are gained through the merging of the wired SDN network domain and the wireless IoT domains.

The significance of this research is its new vision for an efficient resource orchestration by globally connecting local IoT domains over a wired domain managed by SDN technology. The proposed architecture can be applicable for local management of wireless sensor networking (WSN) models such as smart home, smart agriculture, and other domains with the desire for global interoperation for large-scale resource management. In addition, it can enable i) developers to develop innovative IoT applications by leveraging IoT services from multiple IoT platforms; ii) IoT infrastructure owners to gain more interest by sharing their resources; iii) provision of more QoS options for end-users.

This chapter is organized as follows. Section 1.2 provides a brief explanation of terminologies in this thesis title. Section 1.3 indicates the research problems addressed by this dissertation. Section 1.4 states the research aim and objectives. Section 1.5 summarizes the key contributions of this research. Section 1.6 describes the research model and methodology. Section 1.7 presents the structure of this thesis.

1.2 Brief Background

An IoT application mainly demands IoT services from sensors or IoT devices that collect data about the surrounding environment or allow people to control the environment. These devices can be a single body sensor, pieces of smart equipment within a house, a group of sensor nodes over an area like a building, a factory, a garden, a town, or a big city. Due to differences in characteristics of each device as well as the scope of an IoT system that manages the devices, sharing the resources between multiple IoT applications becomes challenging and possibly impractical. To allow numerous IoT devices that are distributed over a large area to share their resources among multiple IoT applications, we need a model that enables the interoperation of separated IoT systems in the provision of IoT services on demand. Moreover, each IoT system needs to be scalable

by itself, so it can easily participate in a large-scale IoT system to share its resources with other applications. The primary focus of this research is to provide a software-based platform that enables an efficient provision of IoT services on demand by orchestrating geographically-distributed IoT systems. This section provides a brief background of the provision of IoT services on demand, software-defined Internet of Things (SD-IoT) platform, and a large-scale SD-IoT platform.

❖ **Provision of IoT services on demand**

IoT services are various in different contexts. IoT services can be defined as follows: “An IoT-service is a transaction between two parties, the service provider and the service consumer. It causes a prescribed function enabling the interaction with the physical world by measuring the state of entities or by initiating actions that will cause a change to the entities” [11]. The IoT services are classified based on the relationship with the entity or the service life cycle. The first approach classifies the services into four types, including low-level service, resource service, entity-service, and integrated service. The second one groups services into 3 categories such as deployable, deployed, and operational. Differently, the services can be i) sensors/IoT devices-associated capabilities such as sensing, processing, storing, actuating, communicating, or connecting; ii) specific requirements for the capabilities. For instance, big data collected from a vast number of IoT devices are defined as big IoT services [12]. In addition, the services can be classified in accordance with the characteristics of service layers [13]. According to the approach, IoT services represent characteristics of each layer in an IoT architecture, for example, sensing data, processed or analyzed data, data related to decision-making, or how services are executed. The prime purpose of IoT applications is to collect meaningful information from the environment in order to take appropriate actions on IoT devices via which we control and manage the surrounding environment. In the proposed platform, IoT services are considered as sensor/IoT devices-associated services such as sensing readings, actuating functions, communicating/computing/networking capabilities, or identification. The services are expected to be provisioned “on demand,” which requires an automatic response as much as possible to any IoT request.

❖ **Software-Defined Internet of Things (SD-IoT) platform**

Software-defined Internet of Things (SD-IoT) platform allows the control, management, provision of IoT services on demand mainly based on the software installed on physical resources as well as resources that are virtualized from the underlying infrastructure such as computing, storage, and networking. The mechanism enables the programmability of networking devices as well as IoT devices using a software entity called the controller. The software controller musters available networking devices and any entities connected to these devices.

❖ **Large-scale SD-IoT platform**

Each SD-IoT system includes IoT devices and a smart management system to control and manage the IoT devices as well as provide IoT services to IoT applications. Each IoT system can be i) directly connected to the cloud, or ii) connected to the cloud via an edge/fog computing system in order to offload computing tasks. A large-scale SD-IoT system is composed of multiple geographically distributed SD-IoT systems connected to the cloud via a core SDN domain. The local SD-IoT systems can be orchestrated to provide IoT services for various IoT demands.

To provision the IoT services to an IoT request over a large-scale IoT domain, the large-scale SD-IoT platform needs to have i) knowledge of requirements of the IoT demand, ii) capability to configure IoT devices to achieve the required services, and iii) ability to program networking infrastructure to deliver results from end IoT devices to IoT users. The provision of IoT services on demand is a capability of instant orchestration of available IoT resources in response to IoT requests. As defined by 5GPP, orchestration presents the automated arrangement, coordination, and management of a complex system, middleware, and services [14].

1.3 Research Questions

IoT can be defined as the “interconnection of sensing and actuating devices providing the ability to share information across platforms through a unified framework, developing a common operating picture for enabling innovative applications.” [15] “How can IoT services on a large scale be customized to the dynamic needs of every user, while

maintaining overall system efficiency and low cost.” [16] However, service-oriented architectures (SOA) still have issues with handling a huge number of IoT devices within an IoT system that exposes unscalable capability [4]. We identified two significant challenges. Firstly, each IoT device and each IoT system need to be programmable, scalable, and manageable for being orchestrated by a large-scale IoT ecosystem. Secondly, it is necessary to control, manage, and orchestrate geo-distributed IoT systems over a wide core network to provision IoT services on demand.


Regarding the first challenge, an IoT system needs to enable the programmability of not only IoT devices but also the network of the IoT devices. In addition, it is essential to be managed by a central entity that allows the IoT system to be scalable to a large-scale IoT system.

As for the second challenge, it becomes necessary to i) manage and control IoT systems as well as the core network connected to the IoT systems, ii) provision the managed IoT sources to IoT applications, and iii) orchestrate the resources in response to IoT requests.

To sum up, the research question addressed in this dissertation can be stated as follows.

“Can the SDN-NFV paradigm be leveraged for orchestration of geo-distributed resources on resource-constrained IoT devices in provisioning IoT services on demand, and can the proposed model be realized in a practical implementation?”

To address the issue, we investigate the following research questions.

 **Can SDN-NFV technologies be ported to the IoT domain but still keep their advantages in the programmability and central management of IoT resources?**

IoT devices as well as IoT systems are limited in communication, computation, power, and storage, whereas SDN-NFV techniques are only applied for powerful devices such as switches, routers or data centers that can handle a massive volume of control messages as well as computation tasks set up by the controller. It is challenging to fully apply the SDN-NFV technologies for constrained IoT devices/networks.

🚧 How can we leverage SDN-NFV techniques in controlling, managing, and sharing IoT resources among multiple IoT applications over limited IoT resources?

It is impossible to fully apply SDN-NFV principles to a constrained IoT system. This thesis aims to respond to this question by investigating virtualization technology for enriching the capability of IoT devices and the Software-Defined Networking protocols for configuring and programming devices.

🚧 How can we automate the resources sharing and orchestration in the provisioning of IoT services on demand over a large-scale IoT ecosystem?

Currently, IoT systems are application-specific. To enable them to share and orchestrate their resources in the provision of IoT services on demand for a number of IoT applications, they need to be scalable and programmable. In addition, we need a large-scale model that enables the local IoT systems to join in or leave the model easily. We address this question by investigating local and large-scale IoT systems, algorithms, and mechanisms for orchestrating, sharing, and managing services on demand.

🚧 How can the proposed models, protocol, and virtual sensors be validated and evaluated in a real deployment?

The proposed model needs to be validated and evaluated in practice. There is currently no standard simulator, emulator, or SDN-NFV-based tools that enable the implementation of SDN-NFV-based approaches to wireless sensor networks or Internet of Things (WSN/IoT) systems. Moreover, there are a few works attempting to implement their proposed solutions, but they fail to provide a complete implementation. To validate the proposed framework, we need to design and develop a software architecture of all components in the proposed model and set up an environment that enables the deployment of not only the SDN domain but also the proposed SD-IoT model. Then, we implement all software components for validation and running test cases regarding demands from IoT applications to evaluate the model's performance.

1.4 Research Aim and Objectives

This research aims to provide a solution to applying the SDN-NFV paradigm for timely and economically provisioning and sharing IoT services on demand by efficiently re-using geographically distributed IoT resources. In particular, the proposed architecture facilitates an efficient and autonomous orchestration of WSN/IoT infrastructure to meet the requirements of IoT applications.

To reach the aims, we define the following objectives and investigate feasible solutions for their achievement.

- ❖ Exploring issues and solutions associated with provisioning IoT services on demand over geographically-distributed IoT systems.
- ❖ Designing a programmable IoT system that can be seamlessly integrated into a large-scale IoT infrastructure to provide IoT services to multiple IoT applications.
- ❖ Proposing a programmable entity that represents an IoT device to enable the IoT device to be programmable and shared between multiple IoT applications.
- ❖ Proposing a control and management protocol that enables representations of IoT devices to be controlled and managed by a central controller.
- ❖ Proposing a programmable IoT framework that allows the control, management, and orchestration of geo-distributed IoT systems to provide IoT services on demand to multiple IoT applications.
- ❖ Evaluating the feasibility and efficiency of the proposed large-scale IoT framework.

1.5 Research Contributions and Significance

This study focuses on the management and programmability of a large-scale IoT ecosystem in the provision of IoT services on demand. Expected outcomes in relation to the above objectives are as follows:

1. A proposed large-scale software-defined IoT framework for orchestrating geographically distributed IoT systems in the provision of IoT services on demand [17]. It enables local IoT systems to join a large-scale IoT system to share their IoT resources for the provision of IoT services to various IoT applications. The novelty of this approach is that it is a paradigm for managing, controlling, and programming not only IoT systems connected to the large-scale infrastructure but also the core network where the geo-distributed systems connect to.
2. A proposed software-defined virtual sensor (SDVS) that is a representation of an IoT device [18]. The SDVS provides advanced capabilities for the constrained IoT devices. It also makes it possible to control both IoT devices and networking of the IoT devices. It enables the programmability of both the functional and forwarding behavior of IoT devices in the provision of their capabilities to IoT applications. With the support of the SDVS, IoT devices can be enhanced with more capabilities and shared between multiple IoT applications.
3. A novel control and management protocol between an IoT device and a local management controller [19, 20]. The protocol allows the controller to program the managed IoT device to provide its IoT services as well as deliver its services to expected destinations.
4. The proposed architecture of a programmable software-defined IoT system can be scalable to be a part of a large-scale IoT infrastructure [21-23]. The system can orchestrate its own IoT resources to provision IoT services on demand. In addition, it can be scalable to be controlled, managed, and orchestrated by a global controller to provide IoT services to multiple IoT applications over a large-scale IoT domain.
5. The proposed framework is implemented for practical realization [17]. A software platform has been developed for validating and evaluating all proposed components of the LSSD-IoT model. The novelty is that the platform provides a software-defined environment for integrating IoT and SDN domains as well as developing SDN-IoT-specific features.

The contributions are of very real significance to not only IoT-associated stakeholders such as service providers, users, developers, but also to a society in general.

For the IoT aspect, the proposed LSSD-IoT model provides the service providers with a management and orchestration mechanism in the provision of IoT services on demand by re-utilizing IoT systems over a wide area. Furthermore, the developers can take advantage of the integrated infrastructure to develop more advanced IoT applications without concern over the deployment cost as well as difficulties in the programmability of IoT systems. Users can obtain more benefits from new IoT applications as well as more demand for IoT services can be achieved.

For the non-IoT aspect, efficient re-utilization of IoT resources enables a reduction in deployment cost as well as minimizing energy consumption for the maintenance of IoT data centers or IoT infrastructure. This would mitigate the impact on the environment.

1.6 Research Methodology

This research adopts the research process and methodology proposed by [24]. This research is divided into four phases, as presented in **Figure 1.1**.

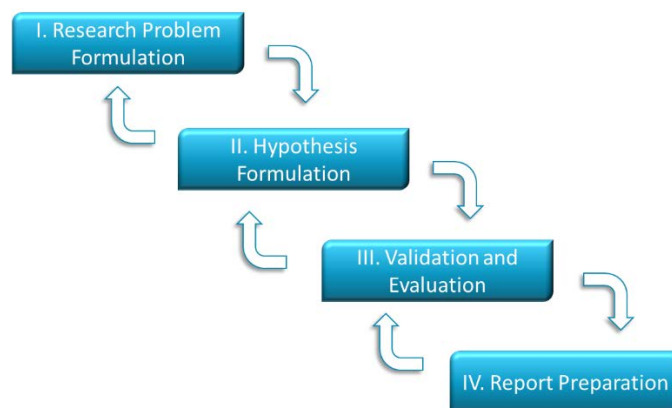


Figure 1.1 Research phases

The research process consists of eleven steps. The first phase is to identify the research problem and review current solutions to the issue. After that, we can obtain a knowledge of the current state of the art about the problem, solutions to the issue, and gaps in the proposals. In the second phase, we develop the research hypothesis and prepare a research plan, including stating the research questions, aim, objectives, and then proposing a new approach to the research problem. The proposed solution needs to be validated and evaluated. In the third phase, we determine the validation approach, conduct it, collect

results from the implementation, and evaluate them to know if we need to refine the proposed model or not. Finally, we write up a report to describe our work and future directions.

Details of each phase are presented in **Figure 1.2**.

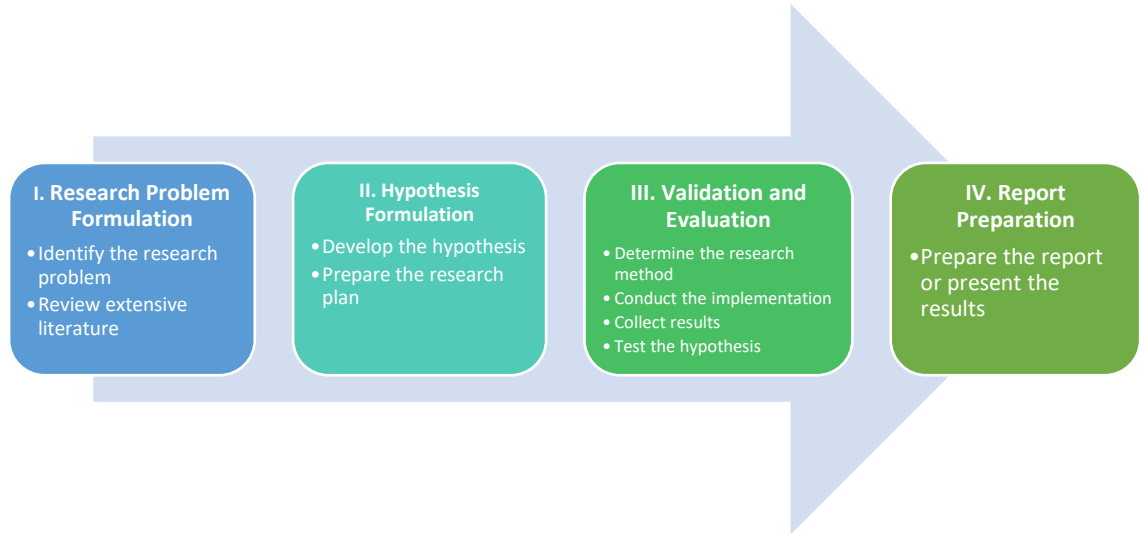


Figure 1.2 Research Methodology

1.7 Thesis Structure

This research has produced one published journal paper, four published conference papers, one conference accepted for publication, and one paper being submitted to IEEE Transactions on Industrial Informatics journal and being under review. The organization of this thesis is composed of eight chapters, as presented in **Figure 1.3**.

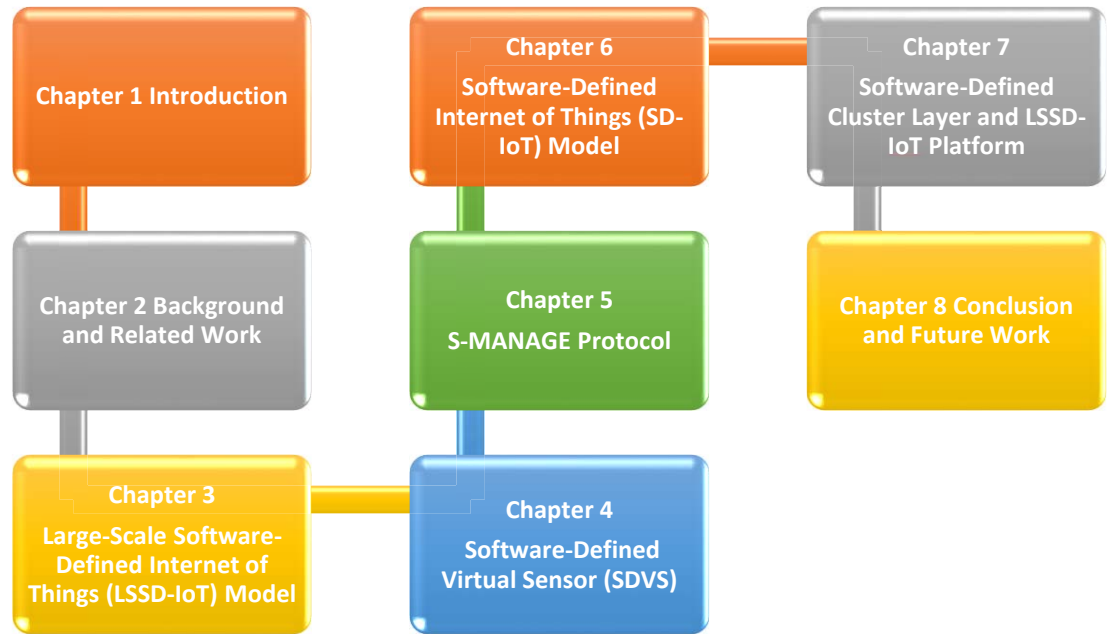


Figure 1.3 Thesis Organization

Chapter 1: Introduction

This chapter introduces an overview of this study. It first provides a brief explanation of the large-scale software-defined Internet of Things (LSSD-IoT) model in relation to the provision of IoT services on demand. It then states the research problem, research aim, objectives, research contributions, research methodology, and provides the thesis structure.

Chapter 2: Background and Related Work

This chapter provides a background of the Internet of Things (IoT) and the provision of IoT services on demand in terms of IoT revolutions stages, enabling technologies and resources for IoT application development, IoT building blocks, IoT architecture and deployment models, and requirements for an IoT system in provisioning of IoT services. This chapter also gives a brief background concerning open-source platforms needed for the practical implementation of the proposed LSSD-IoT platform. In addition, it describes the SDN and NFV techniques used by this research, and challenges of applying the technologies to the IoT domain. It reviews solutions to IoT resource control and management in providing IoT applications on demand, and related work that has applied SDN-NFV in orchestrating WSN/IoT resources. In addition, it presents related works

concerning i) application of virtual sensors, SDN-NFV in programmability of IoT device; ii) SDN-NFV-based proposals for a large-scale IoT architecture; and iii) SDN-NFV-based proposals for a local software-defined IoT architecture.

Chapter 3: Large-Scale Software-Defined Internet of Things (LSSD-IoT) Model

This chapter provides an overall picture of the proposed LSSD-IoT model for provisioning IoT services on demand. In addition, we sketch out contributions of this research in relation to the proposed architecture.

Chapter 4: Software-Defined Virtual Sensor (SDVS)

This chapter presents the proposed software-defined virtual sensor (SDVS). The proposed functional components and software architecture of the SDVS are described. This chapter also provides the implementation results of the proposed SDVS.

Chapter 5: S-MANAGE Protocol

This chapter presents the design and specifications of the proposed S-MANAGE protocol in the control and management of SDVSs for the provision of IoT services on demand. Details of the design and operation of the S-MANAGE are described. In addition, we also present implementation results concerning achieving IoT services on demand via the S-MANAGE.

Chapter 6: Software-defined Internet of Things (SD-IoT) Model

This chapter introduces an overall local SD-IoT architecture in provisioning IoT services on demand. It provides descriptions of the functional components of the proposed architecture. It also presents and discusses the implementation results of the proposed SD-IoT model.

Chapter 7: Software-Defined Cluster Layer and LSSD-IoT Platform

This chapter presents a developed large-scale SD-IoT framework in the provision of IoT applications on demand. It presents the proposed architecture of the SD cluster layer. This chapter also demonstrates the practical implementation of the LSSD-IoT platform and discusses implementation results.

Chapter 8: Conclusion and Future Work

Chapter 8 summarizes this research's contributions and significance and outlines future research work.

Chapter 2 Background and Related Work

2.1 Introduction

This chapter provides a background of the Internet of Things (IoT) and the provision of IoT services on demand. It firstly explores an IoT system in terms of IoT revolution stages together with associated technologies, resources for the IoT application development, key components, architecture, and the requirements for an IoT system. The proposed LSSD-IoT model in this thesis needs to be realized in a practical implementation. Therefore, we review IoT deployment models and real scenarios. In addition, we provide an overview of open-source platforms utilized in this implementation, such as the Floodlight SDN controller, OpenFlow protocol, Mininet, and SDN-WISE framework.

In this research, we apply two techniques, including the Software-Defined Networking (SDN) and Network Function Virtualization (NFV) paradigm in enabling the provision of IoT services on demand. Thus, we provide the background about the two technologies as well as the challenges of integrating SDN and NFV principles into the programmability and management of IoT resources. Moreover, it is necessary to review attempts to overcome the challenges.

The rest of this chapter is organized as follows. Section 2.2 gives an overview of an IoT system in terms of evolution stages, the relationship of things, services and resources, IoT key components, IoT architecture, and requirements for an IoT system. Section 2.3 presents IoT deployment models and real scenarios. Section 2.4 provides a background of the SDN technique and its benefits to the programmability and management of IoT systems, challenges to the integration of SDN-NFV to IoT. Section 2.5 describes the NFV technique and its relation to SDN and IoT. Section 6 describes traditional and SDN-NFV-based solutions to a programmable IoT

device. Section 2.7 reviews works related to SDN-NFV-based approaches to the IoT system's programmability and management. Section 2.8 provides literature on works based on SDN-NFV to develop a large-scale IoT system. Section 2.9 briefly introduces open sources for developing an SDN-NFV-based system. Section 2.10 summarizes this chapter.

2.2 IoT System

One of the indispensable elements in the development of an IoT application is the underlying IoT systems. Each IoT system accommodates numerous IoT devices with various sensing, computing, communicating, and actuating capabilities. It is challenging to control and manage not only the IoT system but also each IoT device in response to IoT applications on demand. First of all, it is necessary to gain an understanding of an IoT system in order to orchestrate it in the provision of IoT services on demand. This section provides a background of an IoT system in terms of evolution in IoT, key components and common features, the definition of things, IoT services, IoT resources and the relationship between them, IoT reference architecture, and requirements for an IoT model in the provision of IoT services on demand.

2.2.1 IoT Evolution Stages

This section provides a brief introduction to the history of the IoT as well as technologies enabling the development of IoT. Definitions and requirements for IoT applications are accordingly changed.

Historically, the "Internet of Things" concept was first coined in 1999 by Kevin Ashton, co-founder and executive director of the Auto-ID Center at the Massachusetts Institute of Technology (MIT). The term was used as a title of his presentation at Procter & Gamble (P&G) to introduce the Radio-Frequency Identification (RFID) technology [25]. From 2000, IoT has attracted much attention around the world with remarkable events, for example, LG announcing a plan for launching an intelligent bridge in 2000, RFID being massively deployed in Savi programs of the American army in 2003, well-known publications such as Scientific American, the Guardian, and Boston Globe citing a number of articles regarding IoT in 2005, and in 2008

the IPSO Alliance being launched by a group of organizations to promote the integration of the Internet Protocol (IP) into intelligent objects and via that to enable the development of IoT. [26]. From 2010, IoT has become popular and remarked with real products and events such as smoke detectors and self-learning thermostats introduced by Nest Labs in 2010 [27], Gartner adding IoT into its famous Hype-cycle for Emerging Technologies list for the first time in 2011 [28], IoT starting to be massively developed thanks to the launch of IPv6 in 2011 [26], and IoT being truly known by the public by the launch of Amazon Alexa in 2014.

The focus of the Internet of Things (IoT) has varied according to advances in enabling technologies. There are three main stages of the IoT evolution [29] , as depicted in **Figure 2.1**. Each stage is characterized by the reference architecture and enabling technologies. In the first stage, RFID and sensors are fundamental elements of IoT. The focus was shifted to internetworking things and web of things in the second generation. Moving to the third generation, the concentration has been put on social, cloud computing and future Internet. IoT is currently moving very rapidly with research regarding Social IoT, Semantic in IoT, information-centric networking (ICN) paradigm in IoT, integration of IoT into the cloud, and RFID-based solutions in IoT [29].

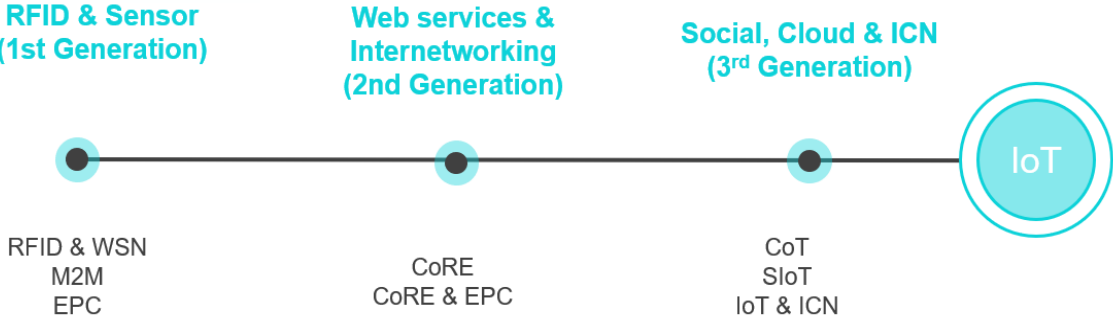


Figure 2.1 IoT Evolution

The development of IoT has involved different technologies. Thus the definitions of IoT have been varied. The following definitions summarized by IEEE in 2015 [30] demonstrate various focuses of the IoT and different requirements for an IoT system with time.

As defined by CASAGRAS in 2009, IoT is “A global network infrastructure, linking physical and virtual objects through the exploitation of data capture and communication capabilities. This infrastructure includes existing and evolving Internet and network developments. It will offer specific object-identification, sensor connection capability as the basis for the development of independent cooperative services and applications. These will be characterized by a high degree of autonomous data capture, event transfer, network connectivity, and interoperability.”

According to CERP-IoT in 2010, “Internet of Things (IoT) is an integrated part of Future Internet and could be defined as a dynamic global network infrastructure with self-configuring capabilities based on standard and interoperable communication protocols where physical and virtual “things” have identities, physical attributes, and virtual personalities and use intelligent interfaces, and are seamlessly integrated into the information network. In the IoT, ‘things’ are expected to become active participants in business, information and social processes where they are enabled to interact and communicate among themselves and with the environment by exchanging data and information ‘sensed’ about the environment, while reacting autonomously to the ‘real/physical world’ events and influencing it by running processes that trigger actions and create services with or without direct human intervention. Interfaces in the form of services facilitate interactions with these ‘smart things’ over the Internet, query and change their state and any information associated with them, taking into account security and privacy issues.”

As defined by IERC in 2014, IoT is defined as "A dynamic global network infrastructure with self-configuring capabilities based on standard and interoperable communication protocols where physical and virtual ‘things’ have identities, physical attributes, and virtual personalities and use intelligent interfaces, and are seamlessly integrated into the information network.". IoT is developed with several technologies that comprise wireless sensor networks, cloud computing, big data analysis, embedded systems, security protocols, security architectures, communication protocols, web services, mobile Internet, and semantic search engines [31].

According to a definition of IEEE in 2015, “IoT will be characterized as a set of interworking networks of things that can be made smart if they can be identified, named, and addressed (smart objects)” [30]. According to the authors, the definition is various in accordance with the scope of

an IoT scenario. Regarding a small IoT environment, “An IoT is a network that connects uniquely identifiable “Things” to the Internet. The “Things” have sensing/actuation and potential programmability capabilities. Through the exploitation of unique identification and sensing from anywhere, anytime, by anything”. The definition has been more complicated when the scope of the IoT environment becomes larger. “Internet of Things envisions a self-configuring, adaptive, complex network that interconnects ‘things’ to the Internet through the use of standard communication protocols. The interconnected things have physical or virtual representation in the digital world, sensing/actuating capability, a programmability feature, and are uniquely identifiable. The representation contains information including the thing’s identity, status, location, or any other business, social, or privately relevant information. The things offer services, with or without human intervention, through the exploitation of unique identification, data capture and communication, and actuation capability. The service is exploited through the use of intelligent interfaces and is made available anywhere, anytime, and for anything taking security into consideration”.

With regard to the involvement of emerging technologies in IoT, IoT today is defined as “a conceptual framework that leverages on the availability of heterogeneous devices and interconnection solutions, as well as augmented physical objects providing a shared information base on a global scale, to support the design of applications involving at the same virtual level both people and presentations of objects.” [29]

As can be seen, the definitions of the IoT are various according to involved technologies as well as user demands. However, they point out some common features of an IoT system [29], IEEE [30].

Interconnection of Things: Refers to interconnecting “things.” Each “thing” is a physical object that is associated with an application or a user.

The connection of Things to the Internet: “Things” in an IoT system are connected to the Internet.

Uniquely identifiable Things: “Things” in an IoT system are uniquely identifiable.

Ubiquity: In the IoT context, ubiquity is an important feature which indicates that a network is available “anytime” and “anywhere.” The “anytime” and “anywhere” is about when it is needed and where it is needed, respectively.

Sensing/Actuation capability: Indicate an involvement of sensors/actuators in an IoT system. Sensors/actuators are attached to the “thing” to achieve sensing/actuation features that make the “thing” smarter.

Embedded intelligence: Is the “smart” or “intelligent” feature of an IoT object that can be seen as an extension to the human mind and body.

Interoperable Communication Capability: The IoT system can communicate with other entities thanks to interoperable communication protocols.

Self-configuration: Programmability: “things” in the IoT can be programmable, which means that the “things” can execute a variety of users’ commands without requiring reconfiguration on their hardware.

Interfaces are necessary connections between things or between humans and things.

Heterogeneity in technologies involved in the deployment of IoT infrastructure demands an interconnection platform that enables the coexistence of these technologies.

Services might be elementary or complicated and are developed from information (sensing, actuating, multimedia, or identification) extracted from each IoT object.

2.2.2 Things, Services, and Resources in IoT

The focus of this research is on the provision of IoT services on demand. However, terms associated with devices/things/objects, IoT services, and IoT resources are still not clearly defined. This section provides the definitions.

In 2010, IETF defined “In the vision of IoT, ‘things’ can vary from computers, sensors, actuators, refrigerators, TVs, vehicles, mobile phones, clothes, food, medicines, books, to people, etc. They can be grouped into three categories, including people, machines (for example, sensor, actuator, etc.) and information (for example, clothes, food, medicine, books, etc.). These ‘things’

need to be identified at least by one unique way of identification for the capability of addressing and communicating with each other and verifying their identities. If a ‘thing’ is identified, we call it as an ‘object’” [30].

In addition, the “things” are defined as IoT devices that have unique identities and are capable of performing remote sensing, monitoring, and actuating [31]. Meanwhile, as specification of ITU-TY.2060 [32], “things” is referred to as an object of the physical world (physical things) or the information world (virtual things); and the IoT device is referred as a piece of equipment which has the mandatory capabilities of communications and the optional capabilities of sensing, actuation, data capture, storage, and processing. Things can be both physical things and virtual things. Physical things can be sensed, actuated, and connected, whereas the virtual things can be stored, processed, and transmitted via the networks. Virtual things can present the physical things via mapping relationships, while there are also some independent virtual things.

The above definitions provide the basic information about “things” (virtual and physical), “IoT devices,” and “object,” but there is still an unclear relationship between “things” and “IoT devices,” “things” and “object,” and questions related to IoT services, IoT resources. The concern is addressed in [33], which provides a clearer definition as follows.

“**Things**” in the Internet of Things represent **physical objects** such as animate objects like humans and animals, and others like cars, machines, consumer goods, rooms, buildings, rivers, or glaciers. The “things” can be seen as the “**entities of interest.**” The “physical object” with *attributes* describing it and the *state* is relevant from the application/user perspective and can be considered as an entity of interest. Hence, “things” and “entities of interest” can be used interchangeably.

“**Devices**” are attached or embedded to the “things” or to the environment where the “things” are monitored. The “devices” are communication elements that enable the “things” to connect to the Internet. Examples of “devices” include RFID readers, sensors/actuators, mobile phones, and embedded computers.

“**IoT resources**” are usually hosted or provided by the “devices.” [29] The resources can be: i) computing and communicating elements, ii) information regarding “thing” as sensing data or

identifier, and iii) actuating capabilities.

“**IoT services**” is an interface through which the outside world can have access to the “IoT resources,” for example, RESTful services. When using REST, “service” refers to application integration and accessing perspective, while “resources” are more about low deployment and low-level components. However, in studies regarding the IoT domain, IoT services are various and classified according to two aspects: i) Relationship with the entity of interest and ii) based on the life cycle [34]. Regarding the first classification approach, there are 4 types of IoT services, including low-level service, resource service, entity service, and integrated service. Meanwhile, the second method introduces 3 types of IoT services such as deployable, deployed, and operational.

As defined by [33] in 2010, a thing is monitored by a device attached to the environment where the “thing” exists or by a device embedded in the “thing.” The “devices” host one or multiple “resources,” which can be accessed via “services.” However, due to the involvement of more advanced technologies with time, the relationship has been expanded to include not only physical IoT devices but also virtual ones, as depicted in **Figure 2.2**. Users can be human or digital users. An IoT application is possibly considered as an IoT service. IoT services interact with IoT gateway, other IoT services, IoT devices, and use IoT data storage. IoT service is an interface between IoT users and other entities. According to the definition of ITU [35], “IoT service provision” involves the activities of using services by IoT customers and providing services by service providers.

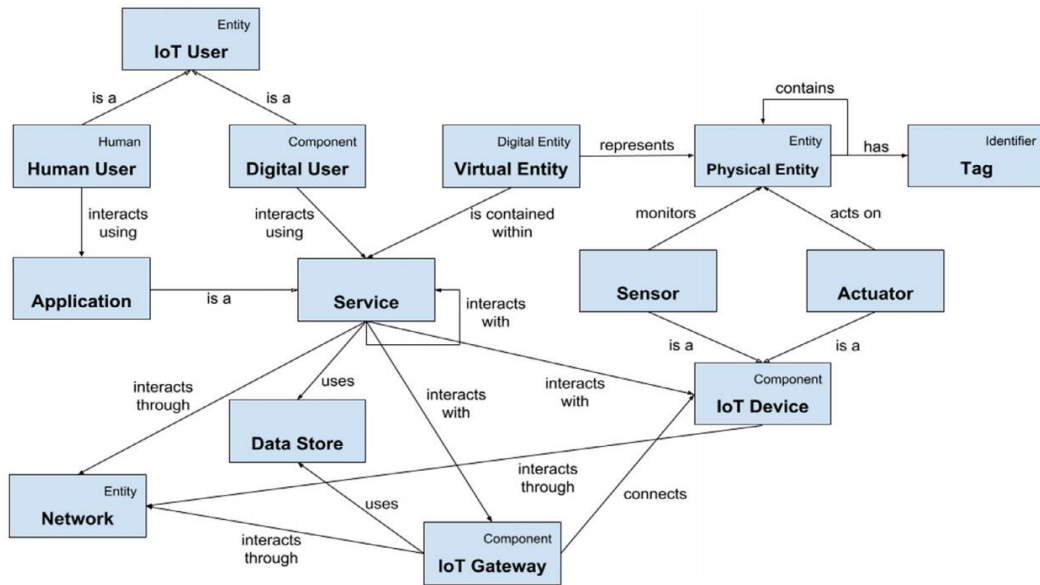


Figure 2.2 Relationship between IoT applications, services, and resources [36]

2.2.3 IoT System – Key Components

To control and manage an IoT system, we need to know what constitutes an IoT system and common features of the system.

The IoT environment is characterized by 6 key components [37], as shown in **Figure 2.3**.

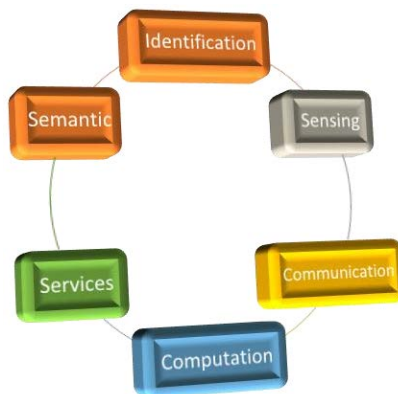


Figure 2.3 IoT building blocks and technologies

Identification

The technique makes it possible for identifying the name and address of smart objects in an IoT environment. Three identification methods are such as object identifiers, communication identifiers, and application identifiers.

Sensing

The sensing component gathers required data from associated objects within the network. The collected data is sent to the database or the cloud, which processes the data as the demand from the associated services. The sensing device can be smart sensors, actuators, or wearable sensing devices, or called “Things.” An IoT device might be composed of several interfaces for communication with other devices, both wireless or wired [31]. The IoT device may consist of 1) I/O interfaces for sensors, 2) interfaces for Internet connectivity, 3) memory and storage interfaces, and 4) audio/video interfaces. The device is capable of collecting a variety of data types from attached or on-board sensors, for instance, light intensity, humidity, or temperature. The sensed data is able to be communicated with cloud-based servers or other devices. The IoT device may be linked to actuators, which enable them to communicate with other physical entities, including non-IoT devices and systems.

Communication

The communication technique is responsible for connecting heterogeneous objects together to deliver specific smart services. Communication protocols used in the IoT environment are composed of WiFi, Bluetooth, Z-Wave, IEEE 803.15.4, RFID, NFC, and UWB.

Computation

The processing units, such as FPGAs, SoCs, microprocessors, microcontrollers, and software applications, represent the “brain” as well as the computational ability of the IoT. A variety of hardware platforms have been developed to run the IoT applications such as T-Mote Sky, WiSense, BeagleBone, Gadgeteer, Raspberry Pi, Intel Galileo, FriendlyARM, UDOO, and Arduino. In addition, to provide IoT functionalities, various software platforms have been utilized. Among the platforms, operating systems are an essential element because they are

always active during the running time of a device. Many real-time operation systems (RTOS) are developed for IoT scenarios. For instance, Contiki RTOS is widely used in IoT scenarios. Other examples can be TinyOS, LiteOS, and Riot OS, which can offer a lightweight OS suitable for the IoT environment.

The Cloud platform is another important computing part of the IoT system. The platform facilitates real-time big data processing that extracts knowledge from the collected big data. The extracted information is eventually beneficial for end-users or IoT applications.

Services

Services can be classified into four categories [38, 39], as in **Figure 2.4**.

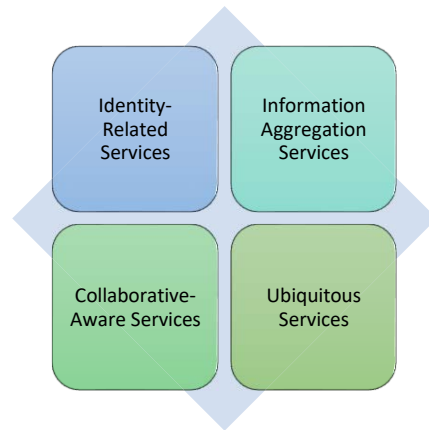


Figure 2.4 IoT service types

Identity-related services are the most important service among the four service types since it is the fundamental service that is utilized by other service types. Every real-world object, that has been brought to the virtual world for developing IoT applications, needs to be identified. *Information Aggregation Services* are responsible for collecting and summarizing raw sensing values that are necessarily processed and reported to the IoT applications. Acting on top of the Information Aggregation Service, *Collaborative-Aware Services* utilize the achieved values for making a decision and reacting accordingly. *Ubiquitous Services* are mainly for providing Collaborative-Aware Services anytime to anyone who has demanded the services.

Ultimately, all IoT applications aim to obtain ubiquitous services. However, it is not easy to achieve this goal because many challenges and difficulties are necessary to be addressed. The majority of existing IoT applications provide the first three service types, including identity-related, information aggregation, as well as collaborative-aware services. For instance, smart grids as well as smart healthcare applications are classified into the information aggregation group. Industrial automation, intelligent transport systems (ITS), smart buildings, and smart home applications fall into the collaborative-aware group.

Semantics

Semantics in the IoT is mainly about the capability to extract knowledge intelligently by utilizing different machines. Knowledge extraction consists of discovering and using resources and modeling information. Also, it includes recognizing and analyzing data for making the right decision on which services are provided for IoT demands. Thus, semantics represents the brain of the IoT by sending demands to the right resource. This requirement is supported by Semantic Web technologies, including the Web Ontology Language (OWL) and the Resource Description Framework (RDF).

2.2.4 IoT Architecture

One of the challenges of IoT resources management is how to ensure the automatic management of resources, different types of IoT architecture, quality of services, and various network infrastructure requirements [40]. Different IoT architectures result in various approaches to resource management. A number of IoT architectures have been proposed according to specific domains such as WSN, RFID, service-oriented architecture, cloud computing, big data, connected living, logistics, smart city, industry, supply chain management, and security [4].

Many attempts have been made to design a common IoT architecture that can meet requirements from industry and researchers. The basic IoT architecture is proposed with three layers [41-43]. Other studies have added more abstraction to the basic structure to include advanced functions of the IoT system. The new model consists of five layers that are named as a middleware-based and SOA-based model [41, 42, 44].

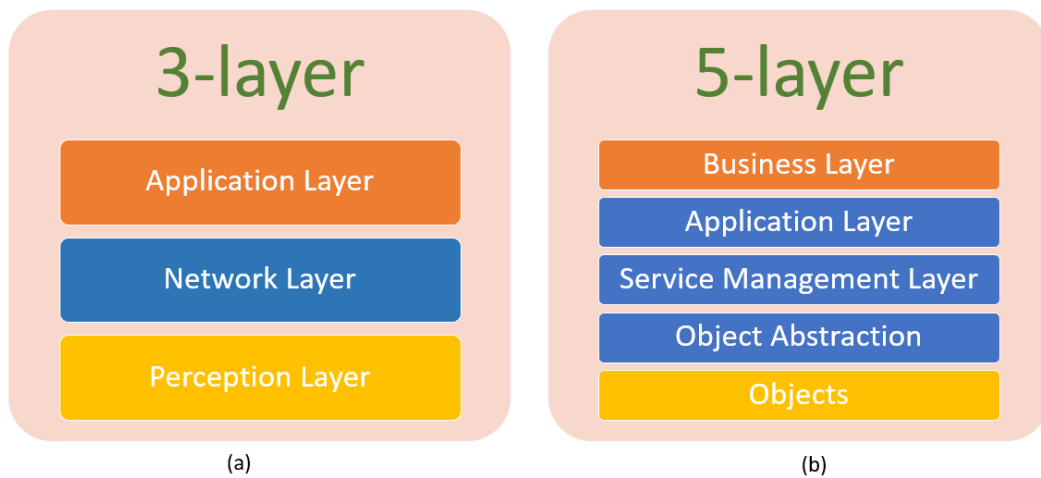


Figure 2.5 IoT architecture

The 3-layer model is comprised of the Application, Network, and Perception Layers (as presented in **Figure 2.5a**). The *Perception layer* is at the bottom of the IoT model. This layer is composed of heterogeneous things, IoT devices, sensors, or actuators that sense the environment, things, or humans. The devices collect data such as humidity, temperature, movement, moisturizer, etc., then digitize and transfer them to the upper layer. The *Network layer* determines the forwarding paths for data obtained from the perception layer and transfers them to an IoT gateway. The *Application layer* is at the top of the IoT model. It integrates the collected data into meaningful information that is provided for IoT applications.

The common five layers of the proposed architectures consist of Business, Application, Service management, Object Abstraction, and Objects Layers, as shown in **Figure 2.5b** [37]. The *Object layer* is similar to the perception layer, as described above. The *Object Abstraction layer* delivers data generated by the Objects layer to the upper layer via a variety of technologies like ZigBee, infrared, Bluetooth Low Energy, Wifi, UTMS, GSM, 3G, RFID. This layer also handles other functions such as data management processes and cloud computing [42]. The *Service Management or Middleware layer* is responsible for pairing services and the user requiring the service. It makes it possible for IoT application developers to interact with the heterogeneity of

objects without concern about their hardware configuration. The *Application layer* provides services required by users. Requested services such as temperature, light, humidity are utilized by various applications such as smart healthcare, building, home and industrial automation. The *Business layer* manages the overall services and activities of an IoT system. It is responsible for building a business model, flowcharts, graphs, etc. in accordance with the data transferred from the Application layer. This layer also designs, analyzes, implements, evaluates, monitors, and develops components associated with an IoT system [41, 43].

The 3-layer model is the simplest one, but the application and the network layer must handle complicated tasks for data processing, forwarding, and management of collected data and required services. Therefore, the five-layer model divides the functionalities and allocates them to five layers.

It is necessary to have an IoT reference model to provide an overall understanding of the primary functions and capabilities of an IoT architecture. The reference model is developed from the 5-layer architecture [32]. This model includes four horizontal layers and associated layers for security and management capabilities (as depicted in **Figure 2.6**). It provides a common view of capabilities and essential functions of the IoT architecture. Moreover, it enables less complexity in the implementation and encourages interoperability among various IoT applications and communication technologies.

- 1) The application layer is comprised of a wide range of IoT applications, for example, smart grid, smart transportation, smart building, or e-health.

- 2) The service and application support layer provides generic and specific support capabilities. In particular, the generic support capabilities can be applied for several applications, while the specific support capabilities can only meet demands from a single application.

- 3) The network layer is composed of transport and networking capabilities. The networking capabilities connect things to the network and maintain the connection. They are responsible for resource allocation, mobility management, routing, or access control. The transport capabilities transport management instructions and IoT application data.

4) The device layer includes a set of gateway capabilities and device capabilities. The device capabilities make it possible for things to directly interact with a network or indirectly through a gateway. The device capabilities consist of ubiquitous sensor networking functions. The gateway capabilities are composed of protocol translation, security, and privacy protection functions to enable resource-constrained IoT devices with heterogeneous wireless technologies, such as ZigBee, Bluetooth, and Wifi, to be connected securely through a network.

5) Security and management capabilities also consist of generic and specific capabilities. The generic management capabilities include device management functions such as remote activation, status monitoring and control, software update, network topology management, and traffic and congestion control. The generic security capabilities include access control, privacy protection, confidentiality, integrity protection.

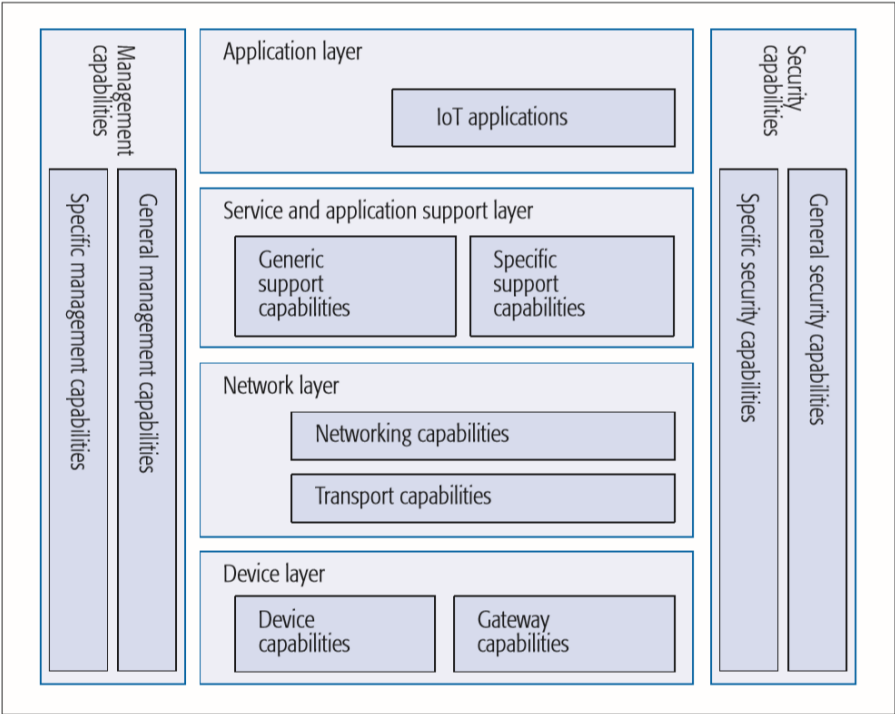


Figure 2.6 IoT reference model [32]

2.2.5 Requirements for an IoT System

Requirements for an IoT system can be classified into two groups, namely functional and non-functional [35]. The functional category is associated with security, data management, devices, communication, service, and application, whereas the other one is relating to operation and implementation. In addition, there are other important IoT requirements, for example, distributivity, interoperability, scalability, scarce resources, and security. They are summarized in **Table 2.1**.

Distributivity: The IoT would possibly evolve in a highly distributed environment. In fact, data may be collected from a variety of sources and processed by a number of machines in a distributed way.

Interoperability: Many vendor-specific devices would necessarily collaborate to achieve common purposes. Moreover, systems and protocols need to be designed in a way that enables devices from various vendors to exchange data as well as to work in an interoperable approach.

Table 2.1 Requirements for an IoT system

Non - functional	Functional	Other
1. Interoperability 2. Operation	1. Application 2. Services 3. Communications 4. Devices 5. Data management 6. Security management	1. Sharable infrastructure 2. Trustable and reliable 3. Service-aware, data-aware, and user-centric 4. Scalable naming and identification 5. Location-independent heterogeneous communication 6. Auto-configurable and remotely controllable 7. Open application programming interfaces

Scalability: The IoT environment is expected to be comprised of a large number of devices. Thus, applications and systems running on top of the devices have to handle an unprecedented volume of generated data.

Resources scarcity: Computation and energy resources would become highly scarce.

Security: Users' being insecure as well as controlled by unknown external entities seriously prevent the development and deployment of IoT.

To meet the additional IoT requirements, some prospective technologies can be utilized, such as SDN-NFV, security/privacy protection, network softwarization, ICN, mobile edge computing, naming, identification schemes, and ID-based communication. **Figure 2.7** [32] illustrates mapping the IoT requirements for enabling technologies. Among these technologies, SDN-NFV can be seen as the potential solution to make IoT infrastructure sharable, remotely configured by dynamically reconfiguring the network, and devices.

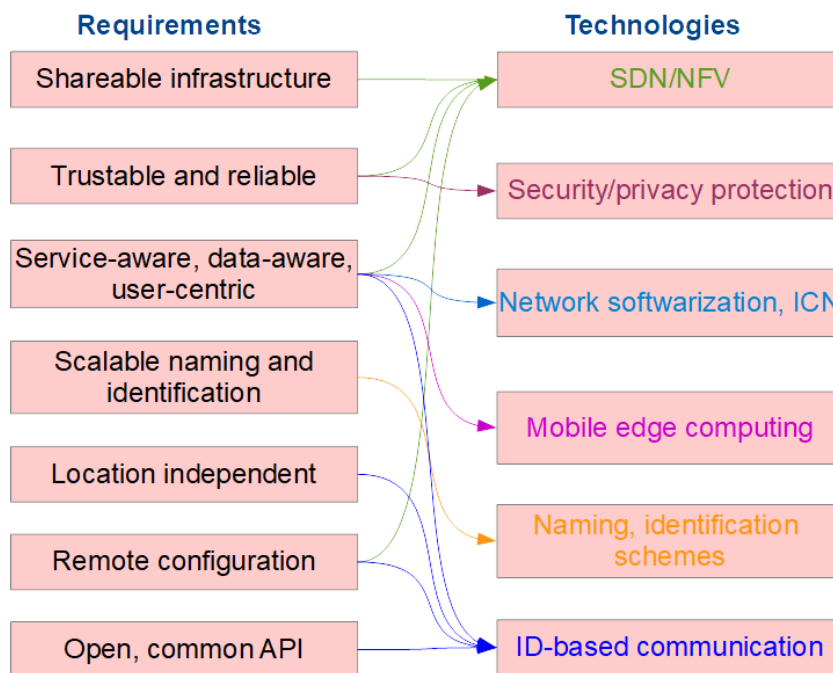


Figure 2.7 Mapping of IoT requirements for enabling technologies

2.3 IoT Deployment Models and Scenarios

IoT deployment models and scenarios are dependent on requirements from IoT applications. This section provides a fundamental model for developing and deploying an application. In addition, we present real IoT deployment scenarios.

2.3.1 IoT Deployment Models

Even IoT applications vary in goals and scope; there are five reference models for the deployment of an IoT application [31].

IoT level-1 system includes one node/device that is responsible for sensing and/or actuating, storing data, performing analysis, and hosting the IoT applications.

IoT level-2 system is composed of a single node that has responsibility for sensing and/or actuating and local analysis, while data is stored in the cloud, and the IoT application is based on the cloud. The system is suitable for IoT applications requiring big data but none of the intensive computations for analysis.

IoT level-3 system comprises a single node for sensing and/or actuating. The cloud stores and analyses data. The IoT application is based on the cloud. The system is appropriate for IoT applications involved in big data and intensive computation for analysis.

IoT level-4 system consists of multiple nodes that are capable of locally analyzing data, while the data is stored in the cloud, and the IoT application is based on the cloud. The model is appropriate for applications that demand multiple nodes, big data, and intensive computation for data analysis.

IoT level-5 system includes multiple nodes and one coordinator node. The end nodes are responsible for sensing and/or actuating. The coordination node performs collecting data from these end nodes and sending the data to the cloud. The cloud stores and analyses the data, and the application is based on the cloud. The model is suitable for applications based on wireless sensor networks, where the involved data is huge, and the data analysis is computationally intensive.

Common components in the models are sensing/actuating element, computing function, data storage, and the IoT application. Differences between them are: i) the number of sensing/actuating elements, ii) position of computing function, iii) with/without intermediate nodes for offloading computing and communicating tasks between the sensing element and the IoT application. The ultimate goal of the IoT system is to be able to be scalable and flexible to manage a number of IoT devices, to distribute computing and communicating functions and to share their resources on demand.

2.3.2 Real IoT Scenarios

In accordance with real IoT systems, [36] has proposed five IoT scenarios, as shown in **Figure 2.8**. These scenarios are derived from IoT applications in smart home and smart city domains.

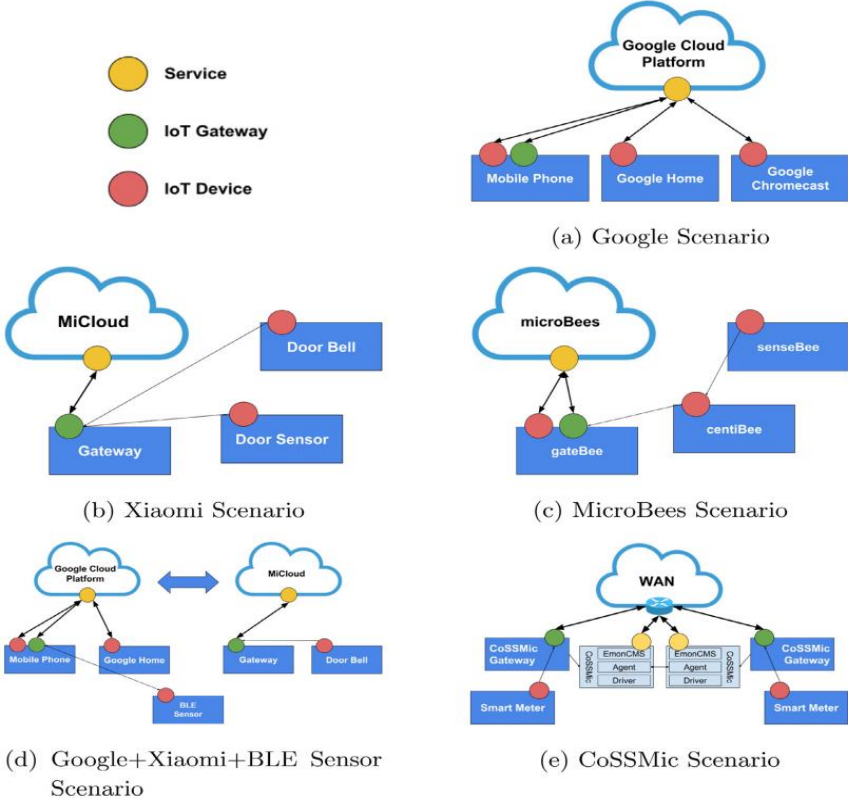


Figure 2.8 Real IoT Scenarios

❖ Smart home scenarios (four cases)

In the scenario (a), the IoT model includes an IoT device and a service deployed on the cloud or called the cloud service. The device is from a specific brand and has an IP address that enables it to directly communicate with the cloud service.

In the scenario (b), the IoT model consists of an IoT device, an IoT gateway, and a cloud service. In this case, the device does not have an IP address, so it communicates with the cloud service via the IoT gateway.

The scenario (c) has similar components, as in the scenario (b). However, in the case (c), IoT devices can communicate with each other via short-range wireless communication protocols such as ZigBee or BLE.

The scenario (d) is the combination of the scenario (a) and (b), where IoT devices with limited communication can also participate in an IoT ecosystem or the cloud.

❖ Smart city scenario: one case

In the scenario (e), each smart home environment becomes a part of a large-scale IoT system where each of them is connected to a WAN. Each smart home system has an IoT gateway that communicates its own resources to other IoT systems via the WAN.

Through the scenarios, especially in the case (e), we can see that there is a need for a mechanism that can provide a distributed as well as central management of not only local IoT systems but also a global IoT ecosystem. In addition, to achieve the aim, each local IoT system needs to be scalable to join the global IoT system on demand.

2.3.3 IoT Application Development - Challenges

Challenges to the development of IoT applications vary according to different layers of the IoT architecture.

Network layer: Challenges to network infrastructure for IoT applications are various since requirements of IoT application domains vary from each other. For instance, reliability and energy efficiency are important to environment-monitoring applications, while QoS is more important to smart grid applications. Multiple optimization technologies have been developed to improve the network performance of IoT networks, such as TCP/IP-based routing, MAC-based technique, MAC-based scheduling technique, and node placement one. To meet specific network requirements of IoT applications, various optimization techniques have been integrated into [45] the network architecture, as depicted in **Table 2.2**. Obviously, the routing technique significantly affects network performance. It is a common technique in the proposed integration approaches. For example, energy and reliability features can be handled by including the metrics in computing energy-efficient and reliable routing paths. Moreover, QoS-related data can be transferred to IoT nodes by including QoS parameters in routing information [45].

Table 2.2 Differences between proposals to network architecture [45]

Applications	Challenges	Integration
E-health	Reliable data transfer	Network layer and TCP/IP layer are integrated
Environmental monitoring	Network lifetime and reliable communication	Network layer and physical layer are integrated
Industrial automation	QoS aware data transfer	Network layer and MAC layer are integrated
Smart grid	Reliability	Network layer and MAC layer are integrated

Sensing layer: According to the research findings from papers published from 2011 to 2017 about deployment and orchestration of IoT over a large-scale infrastructure, most of the approaches fail to consider deployment and orchestration at IoT devices; for instance, there is a lack of consideration of IoT device communication and networking protocols [46]. This becomes impractical since to actually control and manage a whole end-user-to-IoT-device system, there is a need to master low-level IoT devices [47].

Interoperability of IoT platforms: In accordance with a comprehensive review of up-to-date IoT management, there are remaining challenges such as real-time management, interoperability, scalability, security, energy-saving, and performance evaluation [48]. The research in [49] also showed a state-of-the-art of interoperability of IoT systems that current IoT platforms cannot allow more than one IoT platform to be added to an existing IoT ecosystem.

2.4 Software-Defined Networking (SDN) Technique

The Internet has grown into a huge and global interconnecting infrastructure, yet this conventional network is still using complicated and cumbersome network management systems that deal individually and manually with numerous network elements; complex and intertwined distribution of network control and transport protocols; and rigid support for applications and services. In particular, a network device cannot be updated, replaced, or reconfigured easily without affecting other network devices because its distributed control plane and data plane are both embedded in the device itself. New protocols or network architectures cannot be introduced for innovative and emerging applications.

New networking technology is needed, and SDN is not only that technology but also a new networking paradigm because the ideas and principles behind SDN are applicable to the control and management of a system. SDN is attractive to not only academia but also the networking industry with its four key benefits [50] as follows.

- 1) The separation of the control plane and the data plane, allowing them to evolve independently and leaving networking devices simply to forward data efficiently.

- 2) The centralization of network control at a controller external from the network device (the SDN controller or a Network Operating System (NOS)).
- 3) The network programmability via software applications running at the control or application planes.
- 4) The use of flow-based forwarding rules instead of destination-based decisions.

2.4.1 SDN Architecture

As depicted in **Figure 2.9** [51], an SDN architecture is comprised of three main planes, including the application, control, and data plane that accommodate networking applications, controllers, and networking devices, respectively [50, 52].

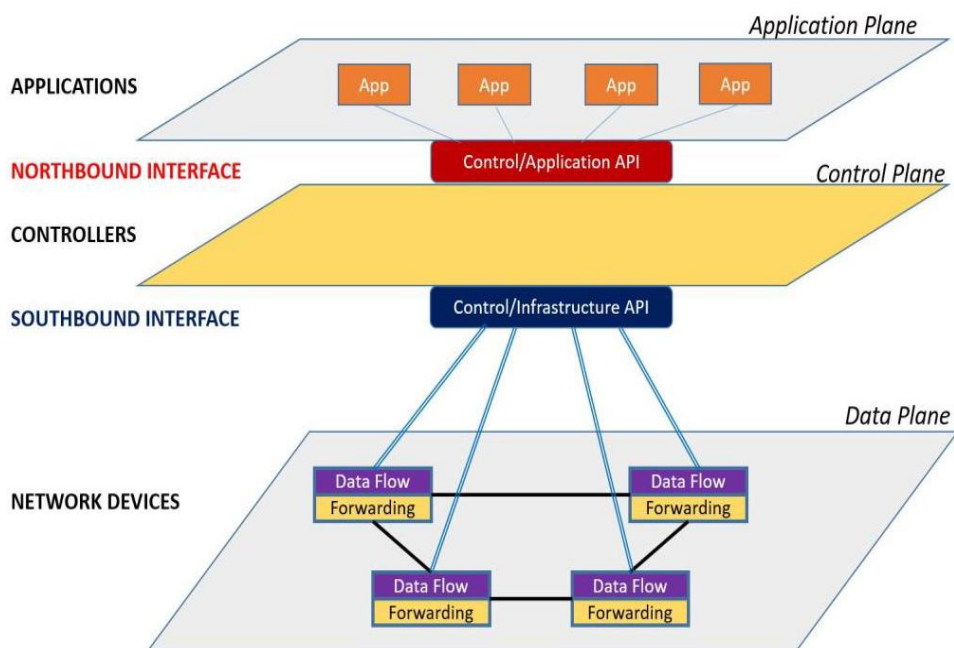


Figure 2.9 SDN architecture

a) Data plane

The data plane comprises both virtual and physical SDN devices, usually known as SDN switches. These devices perform data plane functions and support data plane protocol [53]. In this research, the focus is on the OpenFlow-enabled switches, thus the SDN switch and OpenFlow switch are used interchangeably.

Two primary functions of OpenFlow switches are to support the controller and to forward data [53]. Particularly, SDN switches are responsible for handling and forwarding traffic [54-56] based on the rule information provided by the controller; and gathering network state, temporally storing and transmitting them to the controller [56]. The network status could be network usage, traffic statistic, and network topology.

To handle an incoming packet, the OpenFlow switch has to understand protocol headers in order to extract the required bits, which are necessary for comparing with the corresponding header field in flow table entries [50]. If there is a match, the incoming packet is processed in accordance with the action in the flow entry, and simultaneously, the counter is updated with the statistic of the packet. If no match is found, the packet could be encapsulated and sent to the controller.

SDN devices consist of functional elements such as packet-processing function, an abstraction layer, and an application programming interface (API) to communicate with the controller [54]. The packet-processing function may be the packet-processing software or a packet-processing logic deployed in the hardware that is implemented in virtual switches or physical ones, respectively. The packet-processing function performs actions on incoming packets in accordance with the results of matching the packets against flow entries in flow tables [54]. The abstraction layer abstracts the SDN device as a set of flow tables. In other words, the topology and underlying physical network are not presented to users, while the abstraction is shown as a single router to the user [57]. To allow SDN devices to upward interact with the control plane, the API defines communication approaches, message types, and a secure communication channel between the two entities. **Figure 2.10** presents three main elements of a basic SDN switch, as well as an OpenFlow-based SDN switch or OpenFlow switch.

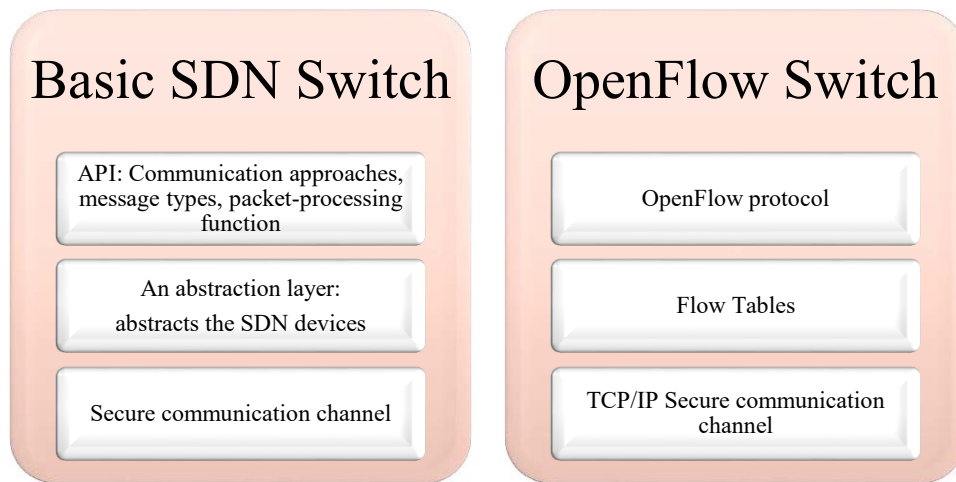


Figure 2.10 Three main components of an SDN switch and OpenFlow switch

b) Control plane

In comparison to the legacy control plane, the SDN control plane is different in three primary ways [58]. Firstly, it can program different data plane elements with a standard programming interface, for example, OpenFlow protocol. Secondly, it exists on a separate hardware device rather than on the forwarding devices, unlike traditional switches where the control plane and data plane are instantiated in the same physical box. This separation is possible because the controller can program the data plane elements remotely over the Internet. The third one is that the controller can program multiple data plane elements from a single control plane instance.

Two main roles of the control plane are 1) managing the infrastructure layer and implementing policy decisions to the data plane via the southbound interface; and 2) providing a global view of the underlying network to the application layer through the northbound interface [50, 52, 59]. The controller utilizes two main parts: a program and the set of rules which are installed on the SDN devices [60]. The program element makes it easier for SDN programmers to only write a specification of intended requests to the network instead of writing the detailed forwarding rules for the underlying network. A compiler is necessary to translate the descriptions into code segments or forwarding rules for the SDN devices and controllers [60].

The SDN control plane is managed by one or multiple central SDN controllers [60]. The controller is considered as a pillar of the SDN architecture owing to its logically centralized control ability and prime responsibility for computation and storage. The performance and scalability of the controller are dependent on the control platform architecture. It is observed that the control platform requires key elements such as i) the applications; ii) the NOS; iii) network abstraction; and iv) communication interfaces or APIs.

The NOS [53] are also called the base network services of the SDN controller. They are all deployed by a set of modules internal to the controller [54]. These functions make it possible for developers to define network policies and manage networks without concern for the details of the network device characteristics, which may be heterogeneous and dynamic [53]. They are discussed as follows. *Network device manager* [50, 54] discovers the state of all SDN devices in the infrastructure layer so it can notify whenever a device joins or leaves the network.

Network device topology management [50, 54] discovers and updates the network changes and then sends the up-to-date network topology to the network applications.

Routing function [50, 54] facilitates the controller to create a route for delivering traffic from a source to a destination.

Flow management [54] : Every controller needs to implement the OpenFlow protocol to perform functions relating to OpenFlow messages, flow table, flow entry, matching, statistics, message queues.

The packet processing unit is responsible for creating appropriate packets for every protocol that the controller can handle. In particular, packets are processed based on their header and payloads. Each packet is composed of source MAC addresses, destination MAC addresses, message types, its parent, and payload. Packets are usually processed for various protocols such as Ethernet, IPv4, LLDP, and UDP.

Security mechanism [50] ensures security enforcement and isolation between applications and services.

The controller platform is mainly represented by the various interfaces. Firstly, a northbound interface directly supports various networking applications that make requests to the underlying network. Another important interface is the southbound interface by which the controller

manages underlying network devices. A controller must implement at least two interfaces [59]. The other interfaces are eastbound and westbound interfaces, which are necessary for communication among controllers and enhance the reliability of the control plane [52].

Northbound interface (NBI): Enables the communication between the application plane and the control plane. It allows end-host applications to autonomously and dynamically send requests to the underlying network. The NBI makes it possible for application developers to control and program the network [52]. It provides the application plane with the abstraction of low-level instructions that support the SBI to configure SDN devices [50]. The NBI is defined as a software system, not a hardware one. Requirements for different network applications are different, so NBIs are various. Currently, there is a wide range of NBIs such as RESTful APIs, Ad Hoc APIs, file systems, and other specific APIs like SDMN API, NVP NBAPI. However, none of them is considered as a standardized NBI [50, 52, 59].

Southbound interface (SBI): Is a bridge between the control plane and the data plane [50, 59, 60]. In other words, it enables the logical link between the SDN forwarding devices and the SDN controller [53]. Noticeably, some SDN controller types may support a single kind of SBI [53]. The SBI offers the interaction method between the control and data elements, as well as a set of instructions for forwarding devices. Some proposals to SDN southbound interface are such as SoftRouter [50, 57], ForCES [50, 55] and OpenFlow [50, 57, 59]. However, they are different in terms of architecture, design, protocol interface, and forwarding model. By comparing their differences and similarities, the authors stated that OpenFlow-based SDN provides higher flexibility and control in terms of development, administration, and network management [59]. Eventually, OpenFlow becomes the most popular standardized protocol [50, 55, 57, 60]. Being the most attractive, OpenFlow-based SDN principles are applied to a number of software-defined wireless sensor network (SDWSN) architectures [61-67].

East-west interface: Enable communication among distributed controllers [50, 52]. Currently, large-scale enterprise networks and data center networks are usually divided into many sub-networks, so a number of controllers are necessary for managing these sub-domains. Therefore, these controllers have to interact with each other to exchange their information related to inter-domain networks and obtain a global view of an entire network. The east-west interfaces

are essential to meet some requirements including advanced data distribution mechanisms; transactional databases; advanced algorithms for fault tolerance and strong consistency; techniques for distributed concurrency; import or export information among controllers; algorithms for information synchronization; notifications of controller capabilities; and backward capability [50].

c) Application plane

The application plane houses network services and applications. Via a high-level programming language in the control plane, the applications can access the global network view and use the underlying services to execute a function [56]. In particular, requirements for an SDN application are defined and translated into commands to program SDN switches [50]. Network services are used to execute network applications and provide them with APIs to communicate with other planes [68]. The applications and services can be deployed within a plane or over multiple planes [68].

The integration of SDN principles and WSN/IoT has been conducted with a wide range of purposes ranging from the idea of how to extend SDN to WSN/IoT to implementation for different research awareness. From the industrial perspective, design of WSN/IoT necessarily satisfies various requirements, including minimal deployment cost and compact size of sensor nodes, energy consumption, quality of service, scalability, multiple sources and multiple sinks, service differentiation, predictable behavior, application-specific protocols, data aggregation, and fault-tolerance [69].

2.4.2 SDN Paradigm's Implications to WSN/IoT

SDN is not just a new networking technology; it is a new paradigm that opens up explorations and solutions in the provisioning and management of resources from infrastructure to applications and services. Logically centralized control allows controllers to gain a complete information base of an underlying network for optional and real-time provisioning of network services. The programmability of the control plane allows autonomous configuration and management of

network devices. The virtualization of resources allows physical resources to be virtualized and support simultaneously multiple services and users.

WSN/IoT does not really operate the same way as switched networks, but they exhibit many similar characteristics. Networked sensors are network devices, and they have to be configured and managed by their controllers. Wireless sensors networks are often organized into clusters and managed by cluster controllers. In a comprehensive application, a wireless sensor network may employ a large number of sensor nodes. Clearly, the above SDN paradigm would bring benefits to WSN/IoT if applied appropriately. Efforts have been made to realize these benefits as follows.

Simple sensor node/IoT device and energy-saving [63, 70]: The device simply forwards data based on the decision of the control logic. The energy consumption of a sensor node is thus reduced.

Routing protocol: Routing technique can be highly improved in the OpenFlow-based SDWSN structure. Yuan, et al. [71] have designed a new routing protocol integrating OpenFlow protocol and a wireless sensor link-state routing protocol, namely Ad hoc On-demand Distance Vector (AODV). The OpenFlow Agent sends less than half control packets per minute and can independently operate when the controller fails. However, the new protocol still accounts for a higher ratio in memory usage and bandwidth consumption. With similar concern, Han and Ren [72] have proposed a new routing protocol in a clustered SDN-based WSN structure. Compared to LEACH, LEACHM, and DEEC, the routing protocol has better performance in terms of the death node number, the network lifetime, and especially has a greater advantage in data transmission. Nevertheless, regarding the load balancing, the new one witnesses the highest rate and thus necessitates a scheme to deal with the issues such as limited battery of center, master and normal nodes, node zoning.

Network management [73], [63]: Network characteristics can be remotely and centrally configured instead of manually and individually reconfigured. Smart network management has been proposed in [63] to address WSN/IoT problems such as power consumption, device mobility, localization and topology discovery, and network management. Nonetheless, the proposal has not been evaluated for its efficiency; the authors suggest future research for

estimating the method's performance regarding reliability and security. Furthermore, the work focuses on the controller architecture without a discussion of the SBI.

Network programmability and innovation: Network policies can be programmed by the software running on the control plane [50, 51] by defining flow entries. This enables a higher degree of innovation in designing network protocols.

Network function virtualization [51]: A sensor node/IoT device can be virtualized to perform the desired network function through a hypervisor in the controller. This opens up a new dimension of services and services provisioning.

Efficient network utilization and services development: With the global view of the underlying network, the controller is able to create virtual networks based on its network abstraction and virtualizes sensor nodes to handle specific-purpose applications. This allows the deployment of multiple WSN/IoT applications over a single physical WSN [64, 74]. Moreover, the controller can flexibly allocate appropriate network resources to an application. This allows the development of infrastructure as a service [16] and platform as a service.

Energy efficiency: This can be achieved with some proposed SDWSN architecture, which enables efficient transmission of packets over WSN/IoT [70], [75]. For example, in [75], an energy-efficient SDWSN-RL prototype based on Sensor OpenFlow [64] has been proposed for environmental monitoring applications.

Cloud integration: The cloud can play an integral part in IoT applications by releasing sensor nodes or IoT devices from the burden of data storage and data processing. With the extension, SDWSN can be developed as sensing as a service.

2.4.3 Challenges of Application of SDN to WSN/IoT

SDN is originally designed for wide area networks with powerful switching and routing devices, so it is difficult to completely apply SDN principles to WSN/IoT because of the constraints of sensor nodes or IoT devices and the wireless medium. Essentially sensors/IoT devices are not switched network devices, and hence, they are limited in their capability.

Furthermore, they do not always use the IP for communications. Developing an OpenFlow-based SD-IoT model may encounter many technical challenges, as discussed below.

a) Designing an OpenFlow-based SD-IoT SBI

Many difficulties are encountered in emulating an OpenFlow-style of SBI because of the difference in the functionality of a switched network device and a sensor/IoT device.

Data Plane – Flow Creation: Typical WSNs employ different addressing as attribute-based naming instead of using IP-like addressing, while packets are processed based on flow entries using IP addresses [64]. This prevents the SD-IoT SBI from creating flow entries, so two methods are suggested: 1) modifying the matching field of flow tables or 2) using uIP/uIPv6 or Blip [64].

Secure channel establishment: A TCP/IP connection between the control plane and the data plane is established based on the IP addresses from two participants in the communication [64].

In-network processing module: It is needed to wirelessly and remotely update sensor firmware and software according to future demands [64]. *Control traffic overhead:* The secure channel can be only hosted with in-band management or out-of-band management in wired networks, whereas only in-band management is supported in wireless networks, leading to the overhead of both data and control traffic [61, 64].

Traffic generation: Traffic is necessarily generated to be adaptable to flow entry definitions of OpenFlow specifications, so a traf-gen module is needed for each sensor node [64].

Power efficiency: It is essential to support duty cycles [76] to periodically turn off the radio, and in-network data aggregation to remove redundant data [64, 76]. This is addressed by an in-net proc module [64] or an additional aggregation player in the protocol architecture of sensor nodes, or new actions in flow tables [76].

Backward and peer compatibility: SD-IoT design is expected to be compatible with traditional networks without SD-IoT SBI or OpenFlow support [64].

b) Designing an SDN-based IoT device

Without changing the basic functionality of a sensor, the challenge is to empower it with adequate capability for software-defined control: a capacity for flow table storage, and capability for handling flow requests from low-tier nodes or the controller. The following aspects should be

taken into consideration in designing an SD-IoT device.

New hardware architecture for sensor nodes may be needed to allow flow-table implementation from the controller. Additionally, memory capacity is sufficient to store the implemented flow tables.

Processing speed: SD-IoT devices have to handle a large number of requests from applications and other nodes. Current switches are unlikely to address flow demands from applications because the SDN devices only forward data and frequently request instructions from the controller to handle arriving packets. This leads to the poor performance of the controller regarding processing power and switch-controller link congestions [60].

Standardized protocol stack: to support network virtualization requires access to heterogeneous sensor nodes or IoT devices to create a variety of virtual networks for the WSN/IoT for serving various IoT applications. Moreover, sensor nodes with different higher protocol layers are difficult to migrate between different networks and communicate with sensor nodes/IoT devices in these networks. Well-defined functional layers are needed to allow sensor nodes/IoT devices to interact properly with the control software.

2.5 Network Function Virtualization (NFV) Technique

This section describes features of the NFV technique and application of the technique in proposing a programmable representation of an IoT device.

❖ *NFV Architecture*

NFV technology allows a pool of physical devices to be virtualized and chained into virtual networking functions that are provisioned as networking services. The main goal is to separate the network functions from physical networking devices. The network function being virtualized is termed a virtual network function (VNF). The VNF is a software instance that can be created without a demand for new physical equipment. Especially, the VNF can be initiated, moved, or terminated on demand [77]. The VNFs are managed and orchestrated by the VNF manager and orchestrator, respectively. As shown in **Figure 2.11**, NFV architecture is composed of NFV

infrastructure, VNFs, and NFV management and orchestration (MANO) elements [78]. The NFV infrastructure (NFVI) provides virtualized resources for building VNFs that is managed and orchestrated by the NFV MANO. Particularly, the NFVS consists of virtual and physical infrastructure. In this layer, the computing, networking, and storing hardware is virtualized into corresponding virtual functions. The NFVI is managed by a virtualized infrastructure manager (VIM) that is responsible for providing virtual machines for developing VNFs. A VNF manager (VNFM) is responsible for initiating, managing, removing, monitoring the operation of all the VNFs. Each VNF can be managed by element management (EM). The VNF can be orchestrated by not only VNFM but also the NFV orchestrator. The NFV MANO can interact with both NFVI and VNFs to orchestrate VNFs.

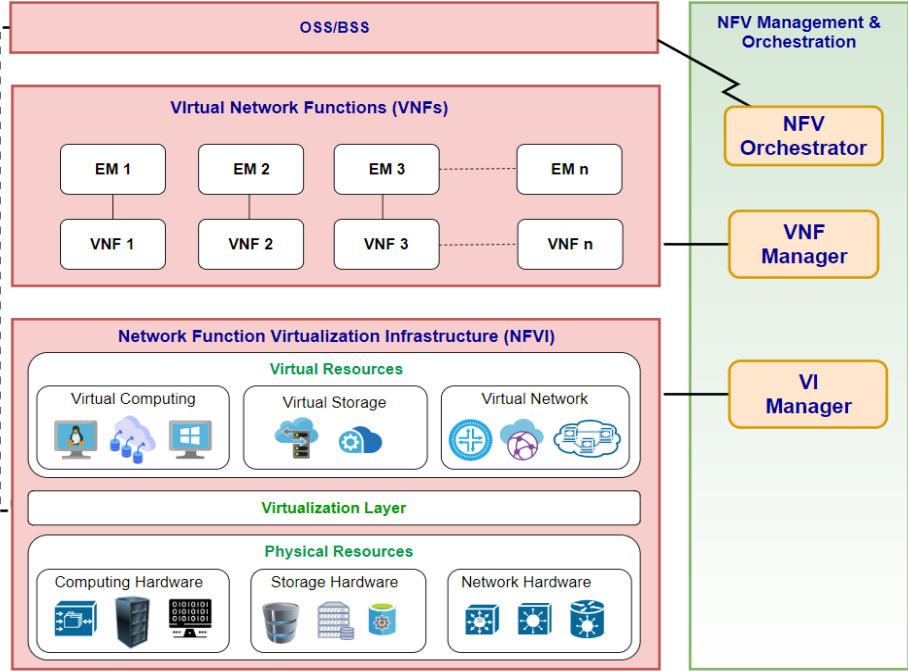


Figure 2.11 NFV architecture

❖ *Relation to SDN*

SDN technology allows central control of networking devices by separating their control logic from physical networking equipment. SDN is comprised of three main components: SDN

applications, SDN controllers, and SDN devices. SDN applications and SDN controllers can be implemented as VNFs, while SDN devices are a part of the NFVI, as depicted in **Figure 2.12** [78].

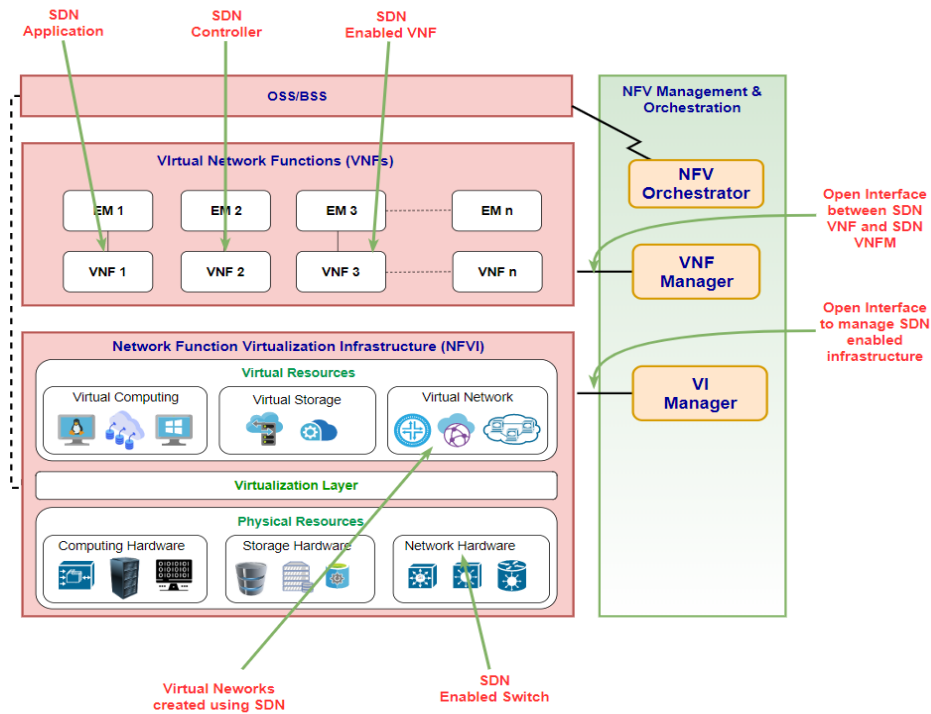


Figure 2.12 Mapping SDN components for NFV architecture

Inspired by the two technologies, we aim to integrate physical IoT resources that could be virtualized and provide a pool of IoT services shared between multiple IoT applications. The physical and virtual resources are controlled and managed by software components.

2.6 Solutions to a Programmable IoT Device

This section provides related work of approaches for proposing virtual IoT devices and SDN-NFV-based approaches.

❖ *Traditional Approaches*

Regarding devices in the data layer, they have many physical limitations: power supply, computation capability, communication protocol, and memory/storage capacity. It is challenging to integrate the SDN and NFV techniques to such physical devices, and various efforts have been made to overcome these limitations. Various types of virtual sensors have been proposed to represent physical devices or their intelligent counterparts. Some provide solutions to IoT issues, including identification, security and privacy, heterogeneity, and scalability [79]. Several types of virtual sensor nodes or virtual objects are developed for WSN/IoT applications. Some emulate a physical sensor to obtain data [80], some implement a software sensor [81]. A software sensor could be an abstraction for providing aggregated data computed from different sensors or supplying predicted or extrapolated data. A virtual sensor may share complex tasks with physical sensor/IoT devices by enhancing and supplementing itself with additional software functions [82]. A virtual sensor may also be a middleware or a software layer on top of a physical device, for example, SenseWrap [83]. A skeleton for a virtual object model can be found in [84]. [85] proposed another version of virtual sensors, Sensor Web, a middleware bridging the sensor resources and applications. [79] discussed a number of proposals for virtual objects for IoT systems. A virtual sensor can serve in various roles. It serves as an IoT gateway to communicate with other components in a network domain, the Internet or a cloud system, using multiple interfaces, possessing protocol conversion capability, and performing management functions.

Nevertheless, there remain many challenging issues [79] that need to be considered for future development of the IoT world regarding virtual sensors and their applications: i) the lack of common association between virtual and real objects; ii) the balanced tradeoff between the number of replicas of the same information and their reusability; iii) the interoperability concern which is the consequence of virtual objects of different IoT systems having different APIs; iv) the scalability issue relating to the management of virtual object life cycle; v) the future creation of virtual objects that may autonomously and adaptively interact with the surrounding environment in order to support dynamic deployment of IoT applications.

❖ *SDN-NFV-based Approaches*

Recently, a number of pieces of research have attempted to leverage SDN and NFV paradigm to introduce new approaches to not only existing but also upcoming IoT issues such as architecture, security, management, programmability and management of IoT infrastructure [86]; or provision of advanced services regarding network virtualization, data distribution, quality of services/experience. However, only a few works consider the benefits of a SDN-NFV-based mechanism in programmability, control, and management of IoT devices in order to provide a complete SDN-NFV-based solution to IoT service provision. [87] has proposed a software-defined device provisioning framework for improving the scalability of IoT platforms, but it fails to consider how to deal with limitations of physical devices in the provision of IoT devices to respond to IoT applications on demand. [88] has proposed an SDN-NFV-based paradigm, that uses virtual images to replace physical devices, for effectively providing collected data to users.

2.7 SDN-NFV-based Solutions to a Programmable IoT System

Efforts to solve WSN/IoT problems with SDN-based solutions have been discussed in [21]. Examples include smart network management of simple sensor devices, energy efficiency in transmission of packets over WSNs, improvement of routing protocols in WSNs, network programmability, and efficient services development.

On IoT network management issues: Several SDN-based approaches have been discussed in [89] in terms of networking perspectives ranging from edge, access, core, to data center networking. They try to bring SDN advantages in networking programmability into configuring networking devices according to the application requirements. [90] contributed to the development of an SDN-based IoT platform by proposing a four-layer architecture, but the architecture allows little programmability of the underlying wireless networks and sensing devices to meet application on-demand requirements. [91] suggested an approach for orchestrating heterogeneous WSN resources to handle various application requirements.

On SDN-NFV-based IoT architecture: Efforts in applying both SDN and NFV techniques to deal with several IoT issues were discussed in [92]. An SDN-NFV architecture was used to take

advantage of virtualization [93]. An SDN-NFV-enabled Edge node for IoT services was proposed in [94] for orchestrating integrated Cloud/Fog and network resources. In [95], an SDN-based IoT framework with NFV functionality was proposed, but the focus was mainly on the importance of distributed OS in the control plane with little concern for IoT device management and programmability.

On challenges of integrating SDN into WSN/IoT: Challenges in integrating SDN techniques into the WSN/IoT environment were discussed in [64, 76]. The main concerns were on components of the application, the controller, and the data layers. Most proposals focused on specific issues of the architecture and inadequately addressed requirements for integration. [96] discussed issues on SDN-oriented architectural requirements for WSNs. [61] discussed issues of multiple controllers, controller placement, and controller core functions. [63] discussed controller core functions and placement. As for the southbound interface, several efforts [64], [76], [65] have been made to modify, extend or adapt the OpenFlow southbound protocol for messages exchange between the control plane and the data plane of sensor/IoT networks. The Sensor OpenFlow [64] extends OpenFlow to deal with the limitations of networked sensors and their specific operating environment. However, Sensor OpenFlow is not practical, and no implementation was found for the proposal. SDN-WISE [65] proposal provided details to the SDN-WISE southbound interface and its implementation performance. [65] also proposed a protocol stack for a generic sensor node and a sink node to communicate with the controller. SD-WISE [97] and Soft-WSN [98] provided a comprehensive description of the design of a software-defined wireless sensor network (SDWSN) architecture. However, the main focus of SD-WISE is on the programmability of a node's forwarding behavior rather than the autonomous configuration management of the node and its functions. On the other hand, Soft-WSN did pay attention to the network and device management aspects for supporting application-aware service provisioning. However, the work presents little consideration for a design of an OpenFlow-based southbound interface for the control and data plane communication.

2.8 SDN-NFV-Based Solutions to a Large-Scale IoT System

SDN has been applied not only in managing wired networking infrastructure but also in addressing WSN/IoT issues in terms of network management and programmability, efficient network utilization, services development, and cloud integration [21]. Together with the SDN technique, NFV has been introduced to address networking-related issues such as network function virtualization. Many efforts have taken advantage of SDN and NFV principles to develop IoT infrastructure for the provision of IoT services over a geographical area. However, they address different aspects of the whole picture that needs to include not only local IoT systems, large-scale management system but also IoT devices.

On edge/fog-computing-based IoT architecture: [8] has proposed a fog of things (FoT) paradigm that joins the fog and cloud computing toward the on-demand Internet of Things. It provides a layer-based FoT architecture with expected features of each layer, but it lacks a performance demonstration. Its contribution is also limited to the scope of the fog network. [99] has proposed a virtualized SDN-based end-to-end architecture for fog networking to manage the complexity of the joint network. Nevertheless, it fails to consider network issues facing IoT systems as well as to provide a performance evaluation of the proposed architecture. [100] proposed a model of integration of IoT, transport SDN, and edge/cloud computing for dynamic distribution of IoT analytics and congestion detection. Being evaluated and validated with jointed experimentation, the work is well-performed in dynamically provisioning IoT analytics between the edge and cloud network and in controlling of bandwidth congestion. However, it lacks consideration of the control and management of data flow circulating within each IoT system, which also contributes to the improvement of data collection in distributed IoT systems.

On network management: [101] developed and deployed an SDN-NFV-based network infrastructure for IoT. It builds and implements applications to slice end-to-end multiple network segments in accordance with the requirements of deploying IoT services from different providers. This work mainly concentrates on core network management without concern for integrated IoT systems or IoT devices. Also focusing on SDN-NFV-enabled network in IoT infrastructure, [102] proposed a two-layer architecture to control network resources in terms of fault tolerance and

load balancing in order to meet requirements of IoT applications, but it limits its contribution to the idea only. [103] proposed a network operating system framework for managing both wired networks and IoT in a unified manner by using the SDN technique. The work has demonstrated the efficiency in the management of networks of switches and networks of sensors via evaluation performance. However, it fails to consider sharing IoT services by leveraging the central management and programmability of the unified network as well as programming functions of IoT devices.

On IoT gateway: [104] proposed a distributed gateway as a virtual network function that is chained in the SDN-based IoT system in large-scale disaster management by leveraging the SDN and NFV technologies. The proposed gateway allows a dynamic integration of distributed IoT devices into a large-scale IoT domain. This work has been evaluated via implementation results. However, it fails to consider the efficiency of the hierarchical management of a large-scale system instead of using a central approach that causes inefficient utilization of the network.

On large-scale system with SDN, NFV and IoT: [105] has proposed a software-defined city infrastructure for integrating owners and administrative domains within a city. It is a two-layer architecture including control and data plane i) for orchestrating, coordinating, and managing the data plane according to specific policies and ii) for programming functionalities of heterogeneous involved devices, respectively. However, the work focuses on the integration of heterogeneity of devices at the data plane and limits their contribution to the idea only.

2.9 Open-Sources for Developing LSSD-IoT Platform

Floodlight controller: Floodlight is one of the well-known open SDN controllers such as ONOS, NOX, Ryu, OpenDayLight, Floodlight, and Beacon. Floodlight has been selected for developing the LSSD-IoT platform due to the following reasons. Firstly, it is a centralized controller, supports the OpenStack cloud orchestration platform, and is designed to be high-performance, which would be helpful for developing a large-scale system. It can handle a mixed OpenFlow and non-OpenFlow network. Secondly, it can provide a module loading system that makes it simple to set up, test, extend and enhance. Thirdly, it is an enterprise-class controller

that is currently developed by Big Switch Network and is well-documented, so the community can achieve great support from the developers of the organization.

OpenFlow protocol: It can be seen as the first SDN communication standard that has been deployed and developed in commercial SDN switches such as HP switches. It allows networking devices to be remotely programmed and managed by a software controller. Through the protocol, the controller can easily update networking devices and configure them dynamically according to networking demands.

OpenFlow Switch: It is an open vSwitch that installs OpenFlow protocol as its Southbound Interface. It enables massive network automation through programmatic extension, while still supporting standard management interfaces and protocols such as 802.1ag, LACP, CLI, RSPAN, IPFIX, sFlow, and NetFlow. Moreover, it is designed to support distribution across multiple physical servers, similar to VMware's vNetwork distributed vswitch or Cisco. In particular, we can deploy it in Mininet for creating a virtual SDN network.

Mininet: It can create a realistic virtual network, running real kernel, switch, and application code on a single machine. It provides an experiment with OpenFlow and SDN systems. It can be integrated with any open SDN controller to build an SDN environment that provides features like a real SDN system.

2.10 Summary

In this chapter, we provided an overview of IoT evolutions and the IoT architectures, building blocks in an IoT system, real IoT scenarios, and IoT reference models for developments of IoT applications. Moreover, a background about SDN and NFV was provided in terms of paradigm, benefits, and challenges of the application of SDN-NFV in IoT. A review of current approaches leveraging SDN-NFV in programmable IoT devices, local IoT systems, and large-scale IoT systems was provided. We also gave a brief introduction to open-sources utilized for developing and deploying the LSSD-IoT platform.

Chapter 3 Large-Scale Software-Defined Internet of Things (LSSD-IoT) Model

3.1 Introduction

Internet of Things (IoT) has developed into an interconnected platform infrastructure for providing essential services ranging from personal health care, smart homes, and smart cities to the manufacturing industry (Industry 4.0), and Industrial Internet. Relying on such an infrastructure, a multitude of emerging IoT services will no doubt be developed for not only local regions but also multiple separated regions spreading over a wide geographical area. To realize these on-demand services timely and economically, programmable and reusable mechanisms are crucial for provisioning and reusing existing resources and infrastructure. Existing IoT services are mostly applications specific, and their supporting infrastructures are rigid and cannot be easily adapted to accommodate new services. This thesis addresses those issues by proposing a large-scale Software-Defined Internet of Things (LSSD-IoT) model for provisioning IoT services on demand with the help of Software-Defined Networking (SDN) and Network Function Virtualization (NFV) paradigms.

The remainder of this chapter is organized into four sections. Section 3.2 justifies the proposed LSSD-IoT model. Section 3.3 describes the LSSD-IoT model. Section 3.4 discusses the features of the LSSD-IoT model. Section 3.5 describes the practical realization of the proposed model. Section 3.6 provides a roadmap of this dissertation. Section 3.7 summarizes this chapter.

3.2 Why Large-Scale and Programmable Services on Demand?

The “Internet” has changed our world and brought with it many technical, economic, and social benefits by connecting people. It is expected that the “Internet of Things” will create enormous value by interconnecting people and everyday things. In fact, IoT has already enabled many emerging applications and services critical to our life in various domains, from personal healthcare to smart cities, critical infrastructures, and supply chain logistics.

However, this enormous potentiality is limited by most existing IoT systems/platforms, which are mainly closed ecosystems since they are vertically developed and deployed in their own IoT infrastructure, and have incompatible standards, formats, semantics, and proprietary protocol and interfaces [2]. As a consequence, we have identified a number of major issues of the current generation IoT systems/platforms:

The astronomical numbers of devices and their connectivity-service infrastructure. As projected, with billions of IoT devices interacting with one another by virtue of their connectivity, the challenge here is how to manage the complexity of the interconnecting infrastructure of these devices and harness their capabilities to effectively serve both local communities and global communities geographically distributed over a large geographical area.

The massive number of IoT services and their provisioning framework. IoT devices are capable of interacting with their environment, performing their designated functions as well as collaborating with other IoT devices. Many services have already emerged to take advantage of these capabilities. The challenge is to automate the provisioning of these services whenever they are needed on demand.

The vast amount of resources and their resource sharing. Collectively, through interconnectivity, IoT systems/platforms present a massive amount of available resources and services to be shared among them. The challenge is in the developing of algorithms and supporting infrastructure for efficient use of the resources through sharing and reusing resources.

In order to address these challenging issues, we investigate technologies, network architectures, device capabilities, protocols, and programmable mechanisms for orchestrating services on demand.

On connectivity and networking architecture: Software-Defined Networking enables network programmability and fine-grained flow-based automated management that are not available with traditional distributed networks. Through the logically centralized knowledge of the whole network, an SDN controller can configure network devices automatically to deal with network dynamics. Many networks are currently being deployed for these purposes [106]. Unfortunately, SDN-like programmability in IoT is still not being commonly used or is still being developed. We aim to adapt SDN for efficient deployment in the IoT domain and investigate a large-scale architecture spanning both the SDN domain and IoT domain.

On device capability: The highly resource-constrained nature of the IoT devices in terms of energy, computing power, storage, and wireless connectivity prevents a direct application of the wired SDN techniques to the IoT world. Many sensor/IoT devices with simple functionality do not possess a programmable interface that is required for resource and service sharing. Research efforts have been attempted to address this issue; systematically enriching devices with resource sharing capability remain open challenges [107]. We investigate the virtualization technology to enhance and supplement the programmability of physical devices.

On communication and management protocol: Sensors/IoT devices are not network routing devices, and heavy protocols for program network flows in network devices are not applicable to IoT devices. Efforts have been made to address this management issue with limited success. We investigate the development of a new simple protocol for adapting and enhancing the SDN paradigm to IoT networks to compensate for the different nature of network devices and IoT devices.

On programmable mechanisms for orchestrating services on demand: We enrich the capability of IoT devices with virtual functions and interface virtualization for orchestrating and programming services. We investigate algorithms and mechanisms for service orchestration.

On resources and services reusing and sharing: IoT services are emerging on a daily basis in many domains; for cost-effective services, both resources and services have to be shared among services and applications. We investigate a programmable platform for sharing the underlying IoT resources and provisioning them on demand. This sharing and reusing of resources and services have not been extensively explored.

The thesis addresses these significant issues by proposing a Large-Scale Software-Defined IoT model and associated techniques for provisioning end-to-end services on demand. It provides a distributed architecture for scheduling, allocating resources for the provision of IoT services. It allows the integration of independent IoT systems (software-defined Internet of Things) to SDN-based systems.

In this chapter, we discuss the significance of the proposed LSSD-IoT model for further adaptation of IoT and future Internet infrastructure. An overview of the proposed LSSD-IoT architecture and its essential components are presented. A roadmap of this research is outlined.

3.3 LSSD-IoT Model

Many IoT-cloud architectures have been introduced to support specific IoT applications that focus on visualization, monitoring, deployment, analytics, data management, heterogeneity management, system management, device management, application development, and research [2], identification, plug and play, search and discovery, quality management, and mobility [3]. As such, they mainly provide application-specific solutions that consequently limit the interoperability of interconnected things.

According to [4], an IoT system can be labeled Internet-oriented, things-oriented (sensors or smart things), or semantic-oriented (knowledge) depending on its intended interconnection, device, or semantic ontology. Generally, the system consists of four key elements: sensor networks with wireless sensors and actuators, machine-to-machine communications, supervisory control, and data acquisition. Features of an IoT system are derived from their constituting elements and are viewed from the application and the infrastructure perspectives [4]. Regarding the infrastructure perspective, there are issues concerning heterogeneous devices, resource-

constrained, spontaneous interaction, ultra-large-scale networks, a large number of events, dynamic and unstructured networks, context-aware, location-aware, and distributed intelligence. Regarding the IoT application perspective, major concerns include diverse applications, real-time requirements, everything-as-a-service (XaaS), increased security attack-surface, and privacy leakage.

Common services required by IoT applications include functional and non-functional ones, as discussed in [4]. It is a considerable challenge for an IoT system to accommodate numerous and various types of services. For example, the design of SOA for service provision has difficulties in handling a massive number of service-based objects since they cause significant overhead regarding data transfer, data process, and management [2].

Networking always presents a big challenge. In regard to scalability, an IoT system needs to scale up in response to the rapid growth of the future increased number of connected devices. Future IoT systems need to support network architectures and network protocols that can be extensible and scalable [5]. It is also necessary to provide open interfaces for future innovation. However, most of current IoT systems are vendor-specific, that expose closed environments. This limits the interoperability and advances in technology [5].

An emerging solution is an application of Software-Defined Networking (SDN) in programming wireless sensor networks or Internet of Things (WSNs/IoT) systems. However, the integration of SDN into the IoT systems exposes a serious challenge owing to the limitation of resources of sensor nodes or IoT devices and the incompatibility of SDN techniques in managing SDN devices and networked sensor/IoT devices.

This chapter describes our proposed Software-Defined Internet of Things model that integrates Software-Defined Networking (SDN) and Network Function Virtualization (NFV) techniques to enable programming IoT devices and network functions of WSN/IoT systems. The system entails a novel model of a Software-Defined Virtual Sensor (SDVS), a streamlined software IoT device (SD-IoTD) controller, and a new and efficient protocol (S-MANAGE) between them for both management and communication. The proposed model allows the programmability of heterogeneous sensor resources for provisioning IoT services on-demand and

their efficient management. A prototype is implemented with reconfigurable software-defined virtual sensors representing their underlying physical/software sensors, utilizing S-MANAGE. The implementation results demonstrate the feasibility and efficiency of the proposed model.

In the context of this thesis, an underlying device refers to i) physical objects such as an IoT device or a sensor node with attached sensors; ii) virtual objects such as virtual sensor nodes, IoT devices, or virtual sensors; iii) a group of physical/virtual objects.

The model involves two levels of management and orchestration (MO). The top MO level (Software-Defined-IoT Cluster) orchestrates on-demand services over multiple clusters over a wide area. The bottom MO level (Software-Defined-IoT Device) orchestrates on-demand services over sensors/IoT devices within a cluster. The architecture entails the design of an IoT-specific SDN controller and a novel Software-Defined Internet of Things (SD-IoT) model. The proposed SD-IoT (SD-IoT) model is composed of elements such as a Software-Defined Virtual Sensor (SDVS), a streamline Software-Defined IoT Device (SD-IoTD) controller, and a new and efficient protocol (S-MANAGE) between them for both management and communication. The proposed LSSD-IoT model allows control, management, and orchestration of not only heterogeneous sensor/IoT resources but also network, data, and applications for provisioning IoT services on demand.

The high-level architecture of the LSSD-IoT model with three principal layers, comprising the application layer, the software-defined cluster layer, and the software-defined device layer, are shown in **Figure 3.1**.

Application layer: This layer consists of end-user applications that utilize LSSD-IoT communications and services. Through the cluster controller, the applications can affect the behavior of the underlying clusters by orchestrating, allocating, and coordinating them to provide the ultimate requested services.

Software-defined (SD) cluster layer: This layer consists of IoT clusters and a cluster controller. Each cluster is responsible for its own local region. Each orchestrates, provisions, and manages services assigned by the controller. Each cluster can be considered as a virtual component that represents the capability of its underlying resources. The controller utilizes the

collective capability of all clusters under its control to provision services requested by the application.

Software-defined (SD) device layer: This layer is composed of IoT devices that are located in widely separated geographical areas. They are organized into groups to form clusters. Each cluster is a platform with its own controller to orchestrate, provision, and manage local services allocated to the cluster based on the capability provided by IoT devices in the cluster.

In summary, an application requests a service from the SD cluster layer, the SD cluster layer orchestrates and distributes sub-services to different clusters depending on their specific capability and location to provision the service requested by the application. Sub-requests are then passed on to the SD device layer. Each platform in this layer is then responsible for orchestrating, managing IoT devices with its cluster to provision the requested sub-service.

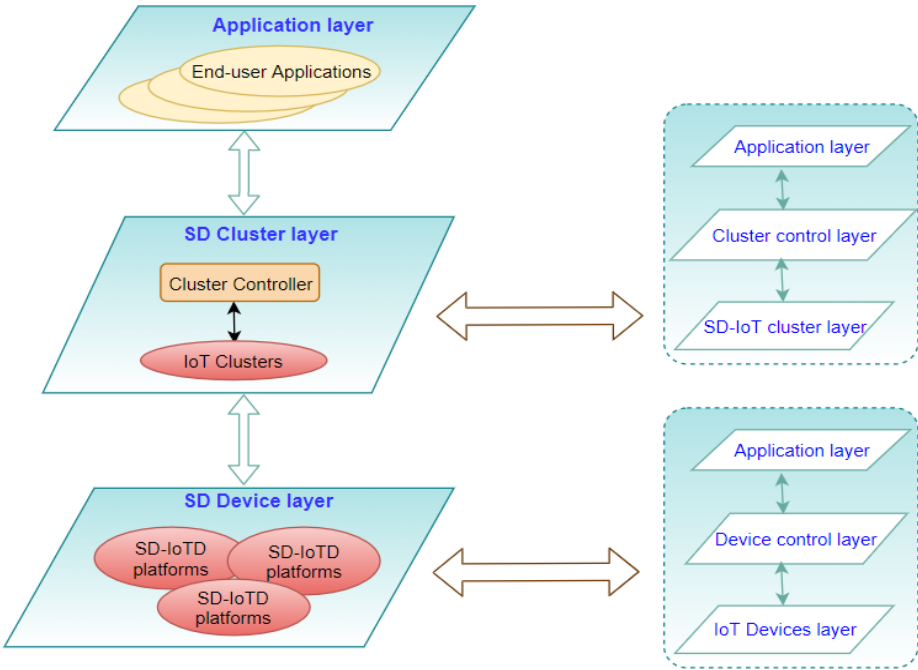


Figure 3.1 LSSD-IoT architecture

3.3.1 Software-Defined Cluster Layer

This layer inherits much of the SDN architecture. It has three layers (**Figure 3.2**): the application layer, the cluster control layer, and the SD-IoT cluster layer. The application layer is the same as the application layer of the overall LSSD-IoT, housing end-user applications. The cluster control layer contains an SD-IoTC controller to perform both SDN functionality and IoT-specific service provisioning and coordinating functions. Instead of just SDN devices, they are replaced by SD-IoT clusters. Each cluster consists of an Open vSwitch and a host that represents an SD-IoT platform below. The OpenFlow and orchestration protocol are used for the communication between the SD-IoTC controller and SD-IoT clusters.

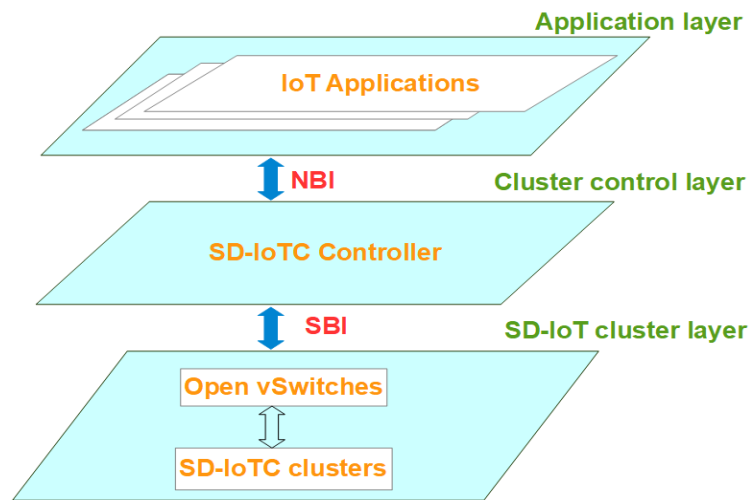


Figure 3.2 SD Cluster layer architecture

SD-IoTC controller: The SD-IoTC controller is a software element that is extended from a well-known Floodlight SDN controller with additional components for managing IoT clusters. It communicates with connected SD-IoT clusters. In addition, the SD-IoTC controller houses a set of components that allow it to i) process IoT requests coming to the LSSD-IoT system, ii) control, manage, and orchestrate IoT clusters/devices, and iii) store temporary IoT services that can be shared between multiple IoT applications. Details of the proposed controller are described in Chapter 7.

Communication interfaces: The communication between the application layer and SD-IoTC control layer is via a Northbound Interface (NBI), e.g., REST-based API. The SD-IoTC control layer communicates with the cluster layer through a Southbound Interface (SBI).

SD-IoT clusters: An SD-IoT cluster is composed of an SDN switch and a host representing an SD-IoT platform. SDN switches are networking devices that connect SD-IoT platforms to the LSSD-IoT system. They report on the connected IoT platforms. They allow the SD-IoTC controller to configure data flows between IoT clusters to deliver IoT requests to a proper SD-IoT platform or transmit returned results to data collection points. Hosts connected to SDN switches are representations of IoT clusters that are composed of required sensors/IoT devices. The hosts can be considered as SD-IoTC clusters that represent the underlying SD-IoT platform.

3.3.2 Software-Defined Device Layer

This layer contains many clusters (SD-IoT clusters). Each cluster is a platform. All platforms have the same three-layer architecture: the application layer, the device control layer, and the IoT device layer (as shown in **Figure 3.3**).

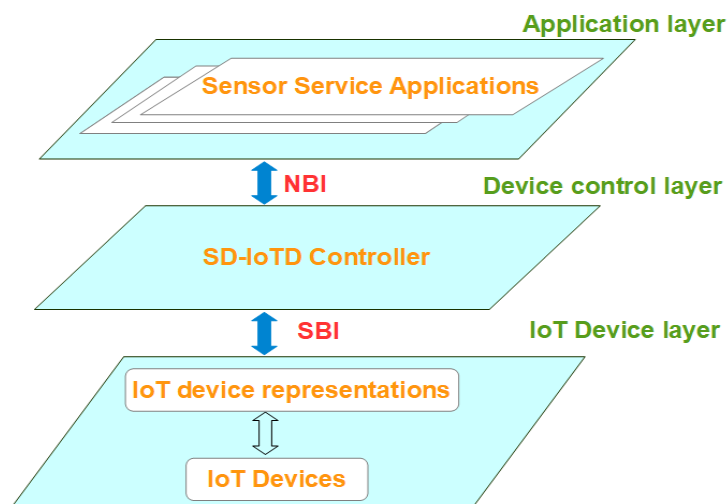


Figure 3.3 SD Device layer architecture

Application layer: This layer allows developers to deploy their IoT applications by utilizing an abstraction of the underlying IoT infrastructure. The abstraction is provided by the SD-IoTD controller in the orchestration layer. The SD cluster layer acts as the application layer to the platform.

Device control layer: This layer accommodates the SD-IoTD controller. It is a bridge between the application and the IoT device layer. It offers the application layer a global view of resources in the IoT device layer. It provides the underlying resources with an interface to update their status, attributes, and sensor services. With the knowledge of both requirements of the application layer and capabilities of the IoT resources within the IoT device layer, it can orchestrate sensor services for IoT applications on demand.

IoT Device layer: This layer hosts SD-IoT resources. It contains IoT devices belonging to the clusters. These devices include both virtual and physical devices (sensors, actuators, cyber-physical systems). We use IoT devices to include networked sensors in this thesis. In particular, virtual devices and IoT devices are software components that can represent plug-in physical devices, emulate physical devices, and logical sensing functions. The description of a virtual sensor or Software-Defined Virtual Sensor (SDVS) is in Chapter 4. Communication between the SD-IoTD controller and the SDVS is through a southbound protocol, S-MANAGE (Chapter 5), which has been designed specifically for communication between the controller and IoT devices.

Different from the SDN data layer comprising robust switches, the IoT device layer is composed of resource-limited IoT devices. Therefore, the device layer is designed with two sub-layers, called representation and underlying layers. The underlying resources consist of physical or virtual sensors/IoT devices, or a group of physical/virtual sensors/IoT devices. The representation layer is an interface between the SD-IoTD controller and the underlying resources. This layer enables enhancements of IoT devices' features, e.g., rigid configuration, limited computation/communication, and networking capability. Moreover, this makes it possible for the SD-IoTD controller to control and program not only forwarding but also functional behavior of the underlying resources for providing IoT services on demand.

3.4 LSSD-IoT Features

This section discusses features of the proposed LSSD-IoT architectures in regard to provisioning IoT services on demand.

❖ Hierarchical Management

In the proposed LSSD-IoT model, IoT resources are under two levels of control and management. The SD-IoTC controller controls and manages SD-IoTC clusters and transporting devices. The SD-IoTD controller controls and manages IoT resources.

IoT resources or IoT infrastructure generally refers to single or multiple physical sensors/IoT devices, or a network of these devices. The SD-IoTC cluster is a representation of an SD-IoT cluster. The SD-IoT cluster consists of software-defined virtual sensors (SDVSs) that represent sensors/actuators or IoT devices. The SD-IoT resources imply SD-IoT platforms and their represented SD-IoTC clusters. The SD-IoTD controller directly controls and manages SD-IoT resources and through which indirectly controls and manages IoT resources. All SD-IoTD controllers are under the control and management of the SD-IoTC controller.

Therefore, the LSSD-IoT system can centrally control and manage IoT resources without direct interaction with them. Through the SD cluster and device layers, the LSSD-IoT system musters the available IoT resources so it can achieve the availability of all integrated IoT systems and orchestrate them to provide appropriate and quick responses to application requests.

❖ Provision of IoT Services on Demand

The provision of IoT services on demand is the ability to instantly respond to IoT demands at any time. Thus, the IoT resource needs to be automatically orchestrated to provision IoT services at any time. According to 5GPPP [14], orchestration describes the automated arrangement, coordination, and management of the complex system, middleware, and services.

There are various IoT service types as discussed in Chapter 2. The IoT services may represent functional elements of an IoT device. The IoT service can be sensing values, actuating status, or an advanced computing/communication capability of an IoT device. One IoT device may provide

various types of IoT services since it may include different sensors/actuators or functions. These IoT services can generally imply data-centric IoT services or function-centric IoT services. The ubiquitous service is the ultimate goal of all IoT systems, and also the LSSD-IoT model.

The LSSD-IoT enables a flexible and scalable provision of IoT services on demand. An IoT service can be established by chaining services of individual IoT clusters in various configurations such as parallel, sequential, or a complex mixture of parallel and sequential configurations. In accordance with IoT application requirements, the LSSD-IoT system orchestrates its underlying IoT resources to achieve the required services.

❖ **Programmability**

The LSSD-IoT enables the programmability of both transporting devices and IoT devices. At the cluster level, the SD-IoTC controller can program the data flows over the core network to deliver IoT requests to IoT clusters and to transport results to desired destinations and configure SD-IoT resources to achieve required services. At the device level, the SD-IoTD controller orchestrates and programs IoT clusters according to the requirements of the SD-IoTC controller. The SD-IoTD controller configures SDVSs and via these programs IoT devices to achieve required services and deliver results to required destinations. The SD-IoTD controller by itself can obtain the availability of the resources and their current service-provisioning tasks, so it can provide appropriate responses to application requests such as meeting the demand fully or suggesting an alternative that satisfies the request partially, or being unable to provide the services because of insufficient resources.

❖ **Virtualization with SDN and NFV**

With SDN: By leveraging the SDN paradigm, the SD-IoTC controller is able to i) obtain a global view of available SDN switches as well as IoT clusters connected to the switches; ii) program data flow between the networking devices in order to forward IoT requests as well as IoT results to desired destinations. Moreover, the principles have inspired the development of a protocol for managing and controlling virtual representations of IoT devices.

With NFV: Using NFV technology, networking functions can be virtualized and hosted by physical servers that are located within Network Function Virtualized Infrastructure (NFVI). With the support of the SDN technique, traffic flows between Virtualized Network Functions (VNF) can be controlled and managed. The SDVS are proposed in accordance with the NFV principles. The SDVS is a virtualized function representing IoT services from an IoT device.

3.5 Practical Realization of the Proposed LSSD-IoT Model

The proposed LSSD-IoT model enables central control and management of IoT devices spread over the large-scale area. The question is whether it is possible to practically control and manage each IoT device in such a large-scale system in the provision of IoT services on demand. In practice, each IoT system is designed to be controlled and managed by each system's own controller, so it is impossible to manage multiple controllers because of their non-interoperability. In addition, with limited capabilities, an IoT device may not be able to handle multiple tasks or be shared among many IoT applications. The IoT devices are mainly application-specific, so it is challenging to reconfigure them without changes in hardware. Therefore, it is crucial to demonstrate the feasibility and efficiency of our proposed model in practice.

Well-known open-sources including Floodlight SDN controller, OpenFlow switches, network simulator Mininet, real sensor tags have been used to demonstrate the practical implementation of the proposed LSSD-IoT platform. The controller of the LSSD-IoT platform is extended and developed from the Floodlight SDN controller to handle IoT devices and their service orchestration. The transport network, representing a large-scale transporting system, includes OpenFlow switches that are deployed in the Mininet setup. The management system of IoT devices is a software entity that can be implemented and modified easily with little changes in the physical infrastructure. The software entity can represent and communicate with physical IoT devices via the device-specific protocol.

3.6 Thesis Roadmap

It is worth noting that this chapter provides the overall architecture of our novel LSSD-IoT model for the provision of IoT services on demand. In order to enable IoT resources to be efficiently utilized by sharing their resources among IoT applications, each IoT system needs to be programmable and scalable to be orchestrated to accommodate new IoT applications, and there needs to be a management model to control and manage these integrated systems. The SDN and NFV paradigms are leveraged by this study.

Even though the SDN and NFV paradigms can meet the demand, there are significant challenges in applying SDN and NFV techniques to the constrained IoT environment. To overcome the difficulties, we propose an SD-IoT model. In the model, we firstly address an issue of constrained IoT devices that cannot be adapted with SDN or NFV technologies. We propose a Software-Defined Virtual Sensor (SDVS) as an enrichment of limited underlying devices. The SDVS enables the programmability, control, and management of the underlying devices in accordance with IoT demands. To control and manage the SDVS, we propose a manage-and-control protocol, termed S-MANAGE. In order to control and manage not only underlying devices but also a whole IoT system in the provision of IoT services on demand, we do need a controller with the capability of controlling and managing their own IoT system as well as sharing their resources with other IoT applications.

A roadmap of the rest of this thesis is organized as follows.

Chapters 4, 5, and 6 describe the design and operation of the proposed SD-IoT model together with novel components, including the SD-IoTD controller, S-MANAGE protocol, and SDVS.

Chapter 7 describes the design of the proposed SD cluster layer and demonstrates the practical implementation of the whole LSSD-IoT platform with all implemented elements, including the SD-IoTC controller, SD-IoTD controller, S-MANAGE protocol, and SDVSs. In addition to the demonstration of the feasibility of the proposed LSSD-IoT platform, we also evaluate the performance of the platform in the provision of IoT services on demand.

Chapter 8 concludes this dissertation by summarizing this study and suggesting future work.

3.7 Summary

This chapter presented the overall picture of our proposed LSSD-IoT model for the provision of IoT services on demand and outlined the roadmap of this dissertation. Firstly, we discussed the need for as well as challenges to effectively utilizing geo-distributed IoT systems to provide services for multiple IoT applications. We then provided a high-level description of the proposed LSSD-IoT model. We discussed features of the model in relation to the provision of IoT services on demand as well as a practical realization of the proposed model. Finally, the roadmap of this thesis was provided.

Chapter 4 **Software-Defined Virtual Sensor (SDVS)**

4.1 Introduction

IoT devices are indispensable elements in IoT systems/services. They provide necessary data or actuating functions for IoT applications (we use underlying devices to include virtual/physical sensors/sensor nodes/networked nodes and IoT devices in this chapter as well as the whole thesis). Deployment of various IoT applications presents many challenges due to their large scale, resource limitation, and heterogeneous environment that accommodates numerous IoT devices with heterogeneous capabilities of sensing, actuating, computing, and communicating [108]. In many existing IoT applications, overlaid deployment of IoT devices causes difficulties in the interaction and sharing of information between the devices and the applications since the IoT devices are too rigid to permit reconfiguration for changes after their implementation. The key challenge is the programmability of various IoT devices in response to diverse IoT application demands.

Among solutions to the programmability of wireless sensor networks and the Internet of Things (WSN/IoT) systems, an emerging Software-Defined Networking (SDN) technique has been proposed. However, applying the SDN paradigm to sensor/IoT networks faces serious challenges due to the limitations on the capability of these devices and their interconnecting protocols [109]. The complementary Network Functions Virtualization (NFV) can partially

address these challenges. NFV technology is utilized to virtualize networking functions as well as enhance the functionality of IoT devices. This technology can be applied readily to the WSN/IoT environment for creating a virtual representation of IoT devices that can serve 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, diverse underlying sensor nodes or IoT devices can be programmed in accordance with the IoT application requests. However, the techniques are only applicable to powerful equipment that has high capability in power, computation, storage, and communication. In order to overcome the challenge, this chapter proposes a Software-Defined Virtual Sensor (SDVS) that provides an enrichment solution for constrained sensor nodes/IoT devices and enables the programmability of the devices in accordance with IoT applications on demand. This work has been accepted for publication [18]. **Figure 4.1** shows the connection between the proposed SDVS and the overall solution of this thesis.

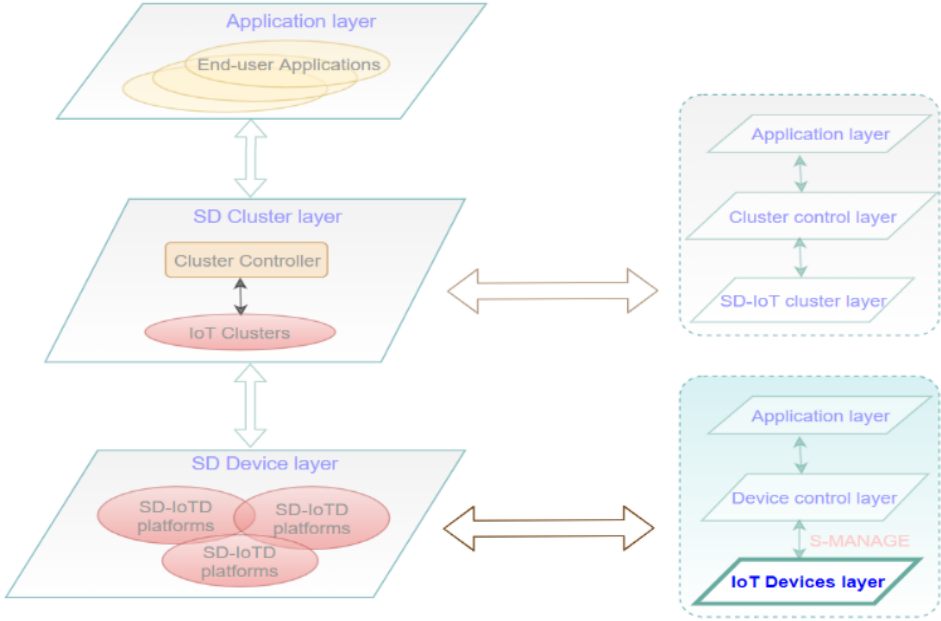


Figure 4.1 SDVS in relation to LSSD-IoT model

The rest of this chapter is organized as follows. Section 4.2 provides a detailed description of the proposed SDVS. Section 4.3 describes the representation types of SDVS. Section 4.4 discusses the features of the SDVS. Section 4.5 presents SDVS architecture and software implementation. Section 4.6 describes the use case scenario and practical implementation of the SDVS. Section 4.7 demonstrates performance evaluation. Section 4.8 concludes this chapter.

4.2 Proposed SDVS

Clearly, an “intelligent entity” has the ability to sense and interact with its environment in a meaningful way in order to achieve the entity’s end goal. A sensor, in its simplest form, is just a physical element or device that transforms some features of the environment into a quantifiable measure for decision making by higher functional levels.

In the Internet of Things, a “thing” is basically defined as an object that can perform a function (or a sensing service), has the ability to connect to a network for collaboration and an identity (addresses or names) so that the device can be called upon to perform its intended service.

A “smart object” is somewhat a glorified term for an intelligent IoT thing, but the term is rather vague without further qualifications as the word “smart” implies a continuum of degrees of intelligence.

In the Industrial Internet of Things (IIoT), the term cyber-physical system (CPS) is defined to mean a system that possesses the capability for sensing and interaction with its environment, the capability for doing computations, and the capability for communicating with other cyber-physical systems.

In its simplest form, a simple sensor is limited in its functionality (simple wires or circuit for sensing the temperature), does not have the capability to perform data preprocessing, or make its output available for further deployment and hence is very limited in its application.

Attempts have been made to create more useful/intelligent sensors. One may surround the basic sensing function with computing and communicating capabilities by embedding in the hardware components such as operating systems and wire/wireless intelligent interfaces.

Other attempts rely on a bare minimum hardware platform and implement all additional intelligent features in software. In other words, with a layer of software over the basic hardware sensor, the composite device acquires intelligence and is able to perform desired functions and to provide services as envisaged.

In many situations, a sensor is purely a software entity (algorithm, object, middleware layer) that either emulates a physical sensor or a software object that provides higher-level services to the user, relying on logical information available within a system.

On computing, additional functionality may include various preprocessing functions that preprocess data (noise removal or compressing) before making it available. On communication, additional functionality may include transmitting and receiving components that can interface with different communication wired or wireless protocols depending on the application. On storage, raw data collected by a sensor may need to be stored locally and temporarily for local preprocessing before transporting to a data collector, depending on the availability of the connectivity at the time.

With the advance of digital transformation, Industry 4.0 specifically, every element of a manufacturing factory has its digital twin as an essential component of the overall smart manufacturing industry. A digital twin of a component is a purpose-built software entity that mirrors the essential features of the component. A digital twin can range from a cyber-physical system (smart sensor, smart conveyor belt, a production line) to a production process or a complete factory. These are emulated software components that an intelligent controller, orchestrator, or an enterprise can use to simulate, test out the operation, and the intended product before committing them to an actual production and manufacturing process.

Many emerging services can now be implemented as cloud services, where a cloud can provide necessary virtual resources on demand (based on a pool of physical resources) and provision services as requested. In this infrastructure of virtualized resources, all computing, storage, and networking resources are implemented in software. Cloud can thus provide services with five defined features: on-demand self-service, rapid elasticity, broadband network access, measured service, and resource pooling.

Clearly, a cloud-based IoT service can be provisioned once elements of the service can be virtualized and orchestrated.

To shield an IoT device from the above-mentioned dependencies, to overcome the limitations of physical sensors/devices, to allow autonomous device configuration and management, and to allow services programmability, in accordance with the SDN and NFV principles, we propose a virtual sensor as a software representation of an IoT device with the following definition and properties.

❖ **SDVS Definition**

A software-defined virtual sensor (SDVS) is defined as a representation and an interface of an underlying device/entity. The underlying IoT resources may include a physical/virtual sensor/IoT device, a set of physical/virtual sensors/IoT devices, a service of a physical/virtual sensor/IoT device, or a subset of services of a physical/virtual sensor/IoT device, as discussed in the Representation section below. It enables a constrained underlying entity to be integrated as an intelligent sensor service in an IoT application that can be provisioned on demand. The IoT services refer to i) sensing data, e.g., light, temperature, humidity, or moisture, and ii) functioning services including communicating, computing, networking, averaging, aggregating, actuating, configuring, or authenticating services.

❖ **SDVS Capability**

SDVS is a software entity that functions as a virtual sensor that addresses the above-mentioned limitations of physical sensors/actuators and possesses capabilities for adapting itself in interacting with the surrounding environment and its controller for providing desired services. Importantly, an SDVS is flexible in communicating with its attached end devices (sensors/actuators) regardless of their specific communication protocols.

Specifically, an SDVS can be considered as an enriched version of a physical sensor or a group of physical sensors by virtue of the additional software layer on top and flexible sets of plug-in sensor interfaces.

An SDVS can provide value-added functions over those available from its underlying sensors through its software implementation. An SDVS may be also able to provide some local processing and management such as data pre-processing and local configuration. An SDVS may also include some storage to deal with other local issues rather than sending them to remote handlers. An SDVS can emulate a sensor, actuator, or provide a digital twin of a device for designing and testing preproduction of service in an Industrial IoT application. By software-defined, we mean that an SDVS can be flexibly designed, programmed, configured, and managed autonomously according to its intended application.

Essentially, an SDVS is defined by its representation type and an interface to its underlying IoT resources.

4.3 SDVS – Representation Types

In accordance with the definition of an SDVS, representations of an SDVS are classified into two main groups: i) sensing and ii) functioning services, as presented in **Figure 4.2**. Regarding the first category, an SDVS may represent a single sensing service that is accumulated from one or multiple physical/virtual sensors/IoT devices. In the second category, an SDVS represents a functional counterpart or an advanced functional element of a physical/virtual sensor/IoT device.

Representing a Physical Sensor/IoT Device: A constrained physical sensor/IoT device may need the additional support of a virtual sensor to become a programmable entity in an IoT system. Particularly, sensing and functioning services of the physical sensor can be programmed to provide IoT services to multiple applications.

Representing a Sensing Service of Multiple Physical/Virtual Sensors/IoT Devices: These provide the same sensing service type. A virtual sensor/IoT device can collect a kind of sensing reading from multiple physical/virtual sensors. The reading can be used for two purposes; aggregating the reading values to produce new average value and deriving a data type that cannot be produced by a single sensing type. For the first purpose, the temperature of small towns is collected for further production of an average temperature of a city. For the second purpose, a

proximity sensor value is produced by interpolating light reading and variance in the light intensity [80].

Representing a Subset of Services of One/Multiple Physical Sensors/IoT Devices: An IoT application may require several sensing service types, and this has to be managed by a virtual sensor through its representation of the underlying sensors. For example, an SDVS may have to abstract different data types from a physical sensor/IoT device and provide aggregated data to the application [81]. Another example in [80], a virtual sensor collects multiple sensor types, such as Temperature sensor and Humidity sensor to compute a safety level value for a Safe habitat monitor or a heat index. The virtual sensor is needed to substitute the sensors that cannot be physically deployed. Furthermore, the virtual sensor is used for informing workers about a safety violation level by computing different sensor types such as temperature, vibrations, and barrels relative proximity [110].

Representing a Subset of Services of One/Multiple Virtual Sensors/IoT Devices: For example, a heat index is derived from moisture and temperature readings. Thus, to evaluate the heat index of a campus, including many buildings, the SDVS necessarily provides average heat indexes collected from multiple virtual sensors/IoT devices that represent heat indexes of buildings within the campus.

Representing a Functional Counterpart of a Constrained Physical Sensor/IoT Device: The SDVS can enhance limited functionalities of a physical sensor, for instance, i) addressing and naming; ii) search and discovery, iii) mobility management, and iv) accounting and authentication. Examples of these cases are discussed in [4].

Representing an Advanced Functioning Service of a Physical Sensor/IoT Device: The SDVS enables an advanced function to be deployed on a physical sensor/IoT device. The additional element enhances the physical sensor's capabilities according to application demands. For instance, an application-specific sensor is used for another application that has a different communication protocol. Thus, the SDVS allows the physical sensor/IoT device to communicate with the application via the required communication protocol.

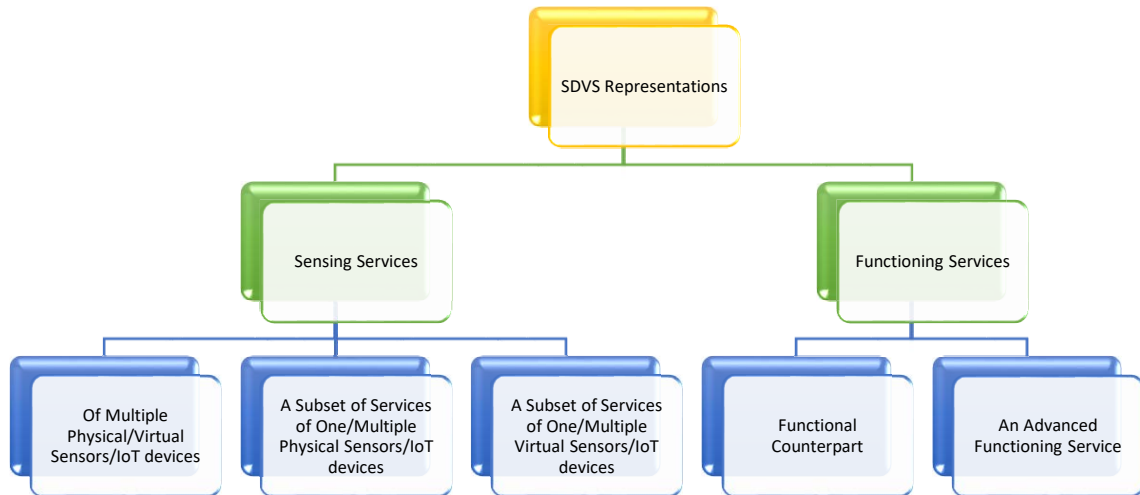


Figure 4.2 SDVS' s representation types

4.4 SDVS Features

To embrace the SDN and the NFV principles and to fulfil the defined representations discussed above, the SDVS is designed with the following features.

Fundamental properties: It has all the characteristics of a represented sensor/IoT device. When representing a service, it provides collected results regarding the service. When representing a physical sensor/IoT device, it has all the characteristics and functions of the sensor/IoT device. When representing a group of physical sensors, it has not only the information of the represented sensors but also additional functions that enable it to handle communication with the physical sensors and high-level tasks. It is also the same as the case of an SDVS representing a group of virtual sensors/IoT devices.

Initiation: It is initiated by a controller when i) an IoT device or a sensor node joins an IoT cluster, or ii) there is a call for a new IoT service.

Location: An SDVS can be placed at the controller or at the physical device according to the resource orchestration approach of the controller.

Activation period: An SDVS is activated according to application demands, so it would be in idle or deleted when it is not involved in any IoT service provision, or its represented sensor no longer exists, respectively.

Identification: Each SDVS is identified by a name, ID, and IP address. A name and ID enable it to locally communicate with each other. An IP address allows it to globally communicate with the controller or applications on the cloud/the Internet.

Configuration management: An SDVS is managed and configured by the controller via a management protocol. The configuration is deployed by the controller and represented by instructions in forwarding and configuring tables. The SDVS follows the instructions to know how to process an incoming packet as well as program underlying sensors, respectively.

Association between the SDVS and physical/virtual sensors/IoT devices: Regardless of many representation types, there are four main kinds of association between an SDVS and underlying devices: i) one-to-one, ii) one-to-many, iii) many-to-one, and iv) many-to-many [111]. As for one-to-one association, an SDVS represents a physical/virtual sensor or a service belonging to the device. Regarding one-to-many association, there are three cases such as i) the SDVS may represent a set of physical/virtual sensors; ii) the SDVS collects information of a homogeneous service from a variety of physical/virtual sensors; or iii) the SDVS represents a subset of services of a physical sensor/IoT device. In this case, one physical/virtual sensor has many services, and each of them is represented by one SDVS, so it results in the many-to-one association. The many-to-many association is a combined method of other associations.

Stored data: An SDVS needs to store information about itself and its represented sensors. The information of the represented sensor includes identification, computing and communicating characteristics, current, past, and future status related to handling tasks and duty cycle modes. The statuses are stores in the list of attributes of the virtual sensor. In addition, it has its own identification, configuration information, including forwarding and configuring tables and statuses.

Communication interfaces: There are four communication types that can be with i) its underlying sensors, ii) other SDVSs, iii) the controller, and iv) other cloud services/applications. It needs protocols to communicate with the controller or other SDVSs, IoT applications, and its represented sensors, respectively.

Security: An SDVS is managed by a controller that is responsible for managing its life-cycle. The SDVS can be created, modified, transferred, deactivated, activated, or deleted by the controller. However, the controller can delegate the control to an application/user, another controller, or other SDVSs with limited permission.

Mobility: An SDVS can be moved between IoT systems or within an SDN domain, or to the cloud.

Service advertisement: An SDVS informs and updates its available services to the controller on behalf of its represented sensors/IoT devices.

Service provision: An SDVS stores temporary sensor services collected from its represented sensors and reuses them to quickly respond to application requests.

Programmability: An SDVS can be programmed to operate according to its configuration set up by the controller. It is responsible for communicating with represented sensors via their specific protocol.

IoT Interface: An SDVS is an interface between underlying IoT resources and the controller, applications/users, or the cloud.

4.5 SDVS Architecture and Software Implementation

4.5.1 SDVS Architecture

Typically, a physical sensor node is composed of four units: computing, communication, sensing, and a power subsystem [112], as shown in **Figure 4.3**. The computing subsystem controls the elements of the sensor node and computes required tasks [112]. It includes a processor and a storage unit. The processor unit may operate in various energy-saving modes as

Sleep or Off-Duty, Active or On-Duty, sensing unit on duty, and transceiver on-duty mode. The storage unit comprises a flash memory, containing the program code for a node, and a RAM, storing sensing data and necessary data for computing tasks. The communicating subsystem allows a node to communicate with other nodes or with the base station by using a short-range radio. The power subsystem includes a battery. The sensing subsystem translates physical phenomena into electrical signals.

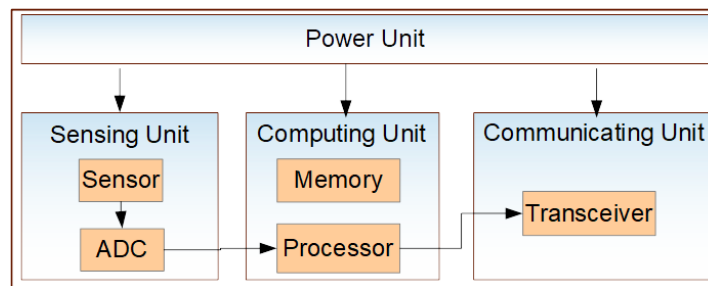


Figure 4.3 Sensor node architecture

However, depending on their applications, underlying devices may have different functionalities and deploy various technologies. In terms of communication, they may use different wireless mechanisms that entail different routing protocols, addressing schemes, data encodings, and data formats. In terms of IoT platforms, they may rely on different development environments, programming languages, processors, memories, and communication networks [113]. IoT devices can be of various types, but they typically include common elements such as i) identification, ii) sensing and actuating, iii) computation, iv) communication interfaces, and v) management. They may consist of several communication interfaces such as i) audio/video, ii) memory and storage, iii) Internet connections, and iv) I/O interfaces for sensors.

To represent such basic devices as well as fulfill the proposed features, the SDVS architecture requires three main elements, as depicted in **Figure 4.4**.

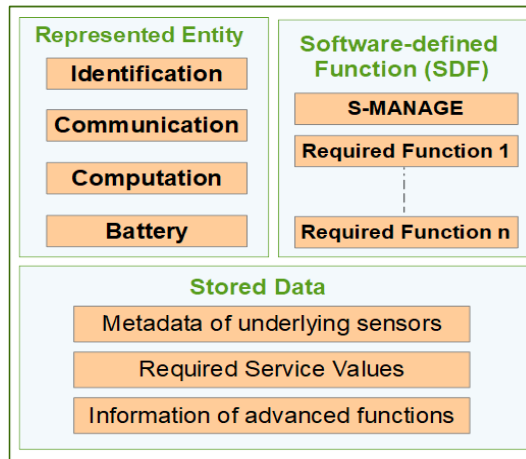


Figure 4.4 SDVS architecture

Represented entity: It includes attributes and functions of the represented underlying entity, for example, battery level, driver communication protocol, attached sensors, processing capability, memory.

Software-defined functions (SDF): They are communicating or computing functions that can be programmable via a management protocol.

Data storage: It stores metadata of the represented entities, sensing data, and instructions for configuration of represented entries or for forwarding results to desired destinations. Particularly, it temporarily stores actual data collected from its represented entities.

4.5.2 Software Implementation

To perform the proposed functions, the SDVS is composed of four main software components (as illustrated in **Figure 4.5**), including i) identification and address for communicating with both controller and represented entities, ii) sensing/actuating services of represented devices, iii) storage data element storing information about sensing/actuating services, and iv) advanced functions that can be flexibly installed such as communicating and computing functions.

```

"SDVS":{
  "identification": {
    "Name": "SDVS01",
    "Net": "0.0",
    "IP_Addr": "0.21",
    "MAC_Addr": "00:00:00:00:00:01",
    "Dpid": "000001",
    "Lis_Port": "7777",
    "LOC": "LOC01",
    "state": "0"
  },
  "representation": {
    "SensorList": "sensorlist"
  },
  "communication": {
    "to_controller": {
      "Name": "localhost",
      "Port_Num": "7779",
      "Type": "S-MANAGE"
    },
    "to_SDVS": {
      "Type": "S-MANAGE",
      "Connection": "none"
    },
    "to_representeddev": {
      "Type": "physical_sensors",
      "Driver": "BLE"
    }
  },
  "computation": {
    "Type": "Aggregation"
  },
  "storage": {
    "Type": "SDVS01_Mysql"
  }
}

```

Figure 4.5 SDVS in software

To communicate with SD-IoTD controller via S-MANAGE protocol, the SDVS is implemented with an algorithm (**Algorithm 4.1**) that assists the SDVS to process an incoming packet. Each packet header is extracted from an incoming packet and is stored as a set of header fields. If the flag (first bit of Type) value is 0, the action is to point to the Forwarding Table to get further instruction for forwarding the packet. If the flag value is 1, the action is to point to the Controlling Table to get rules related to the configuration of underlying sensors. A set of incoming packets to the SDVS is processed to get the header fields (line1-2). Analyzing the flag value in the header field is used to select the right table to get further actions on the packet (line 4-10).

Algorithm 4.1: SDVS's handling packet

Input: A set P of packets.

Output: returned a set of action

```
1: for each  $p$  in  $P$  do
2:    $h = p_{header}$  //  $h$  is a set of packet header fields
3:   if  $h_{flag} = 0$  then
4:      $h_{action} = Select(Forwarding\_Table)$  //get the right action specified in this table
5:     execute ( $h_{action}$ )
6:   else
7:      $h_{action} = Select(Controlling\_Table)$ ; //get the right action specified in this table
8:     execute ( $h_{action}$ )
9:   end if
10: end for
```

The functional components of the SDVS are defined by software classes. They are written and implemented in Java. All the classes enabling the operation of the SDVS and relationships between the classes are illustrated in **Figure 4.6**.

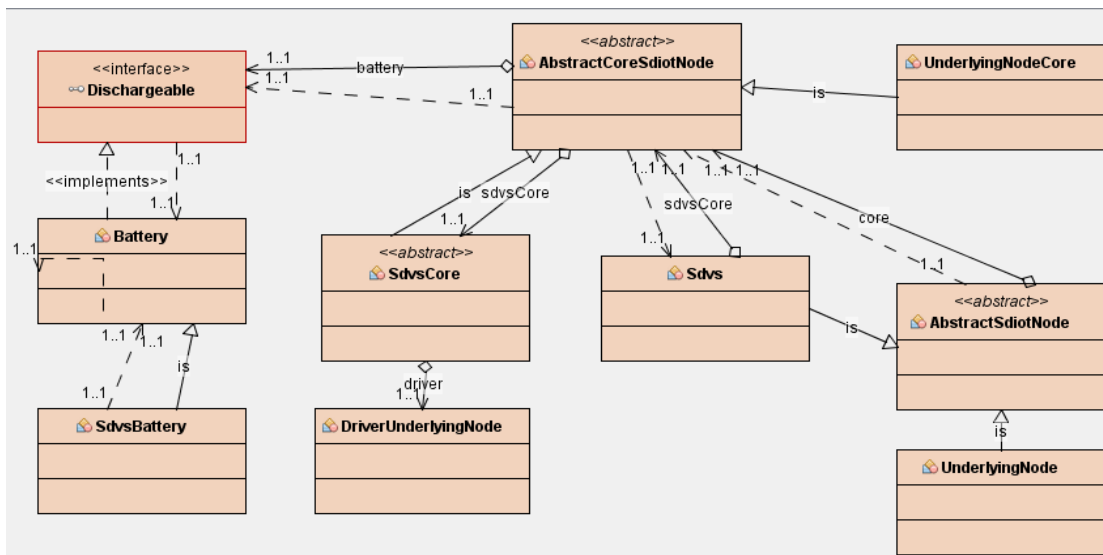


Figure 4.6 Class diagram of the SDVS

The above classes defined attributes and functions of the SDVS. An SDVS needs to have fundamental attributes of sensor nodes or IoT devices, and advanced functions as specified in its architecture. To represent an underlying sensor node/IoT device, the SDVS needs to have the represented entity’s information concerning battery (Battery.java and Dischargeable.java, SdvsBattery.java), driver or communication protocol (DriverUnderlyingNode.java), fundamental parameters and functions (UnderlyingNode.java).

The SDVS is designed to be able to communicate with a controller via S-MANAGE protocol and with an underlying represented entity via its device-specific protocol. SDVS needs to be identified by ID number, address, port number, its location. Moreover, it has information about the underlying node, including identification, driver, attached sensors, or capabilities. It also knows its neighbor sensor nodes. All parameters constituting the SDVS are presented in the SDVS java class, as shown in **Figure 4.7**.

```

class Sdvs {
    -String sdvsId
    -String locID
    -String state
    +NodeAddress sdvsAddr
    +AttachedSensor attachedSensor
    +List<AttachedSensor> attachedSensorList
    -InetSocketAddress ctrl
    -DataInputStream dis
    -DataOutputStream dos
    -Socket tcpSocket
    -AbstractCoreSdiotNode sdvsCore

    +Sdvs(byte net, NodeAddress myAddress, int port, InetSocketAddress controller, String neighboursPath, String logLevel, String dpid, String mac, long sPort, String sdvsId, String locID, String state, List<AttachedSensor> attachedSensorList)
    +String getSDVSid()
    +NodeAddress getAddr()
    +NodeAddress getSDVSAddr()
    +String getLocID()
    +String getState()
    #void startThreads()
}

```

Figure 4.7 SDVS’s java class

The fundamental parameters of an SDVS, as well as the underlying node, are defined in the AbstractCoreSdiotNode.java class (**Figure 4.8**). The class also contains abstract methods that represent the fundamental functions of an SDVS and its represented entity (**Figure 4.9**).

```

<<abstract>>
AbstractCoreSdiotNode

- List<AttachedSensor> attachedSensorList
+ SdvsCore sdvsCore
+ final byte NULL
+ final byte Y
+ final int FUNCTION_HEADER
+ final int MAX_RSSI
# final int QUEUE_SIZE
- Dischargeable battery
- int cntBeacon
- byte requestId
- int sdvsDistance
- List<NodeAddress> acceptedId
- List<FlowTableEntrySdiot> flowTable
- List<ConfiguringRuleEntrySdiot> configTable
- List<ForwardingRuleEntrySdiot> forwardTable
+ ArrayList<ConfigRuleEntry> configTableInByteArr
+ ArrayList<ForwardRuleEntry> forwardTableInByteArr
- ArrayBlockingQueue<NetworkPacketSdiot> ftQueue
- HashMap<Integer, LinkedList<byte[]>> functionBuffer
- HashMap<Integer, FunctionInterfaceSdiot> functions
- boolean isActive
- ArrayBlockingQueue<Pair<Level, String>> logQueue
- NodeAddress myAddress
- int myNet
- Set<NeighborSdvs> neighborTable
- int rssiMin
- int ruleTtl
- ArrayBlockingQueue<Pair<NetworkPacketSdiot, Integer>> rxQueue
- HashMap<String, Object> sensors
- ArrayList<Integer> statusRegister
- ArrayBlockingQueue<NetworkPacketSdiot> txQueue
~ int countDataPktGet
~ int countAckSetOn
~ int countAckSetOff
~ long dstRxTime
~ int countSetFwdRulePkt
~ int countSetConfigRulePkt
~ String newStatus

```

Figure 4.8 Basic parameters defining an SDVS and its represented entity


```

~AbstractCoreSdiotNode(byte net, NodeAddress address, Dischargeable bat, List<AttachedSensor> sensorList)
+ Dischargeable getBattery()
+ int getFlowTableSize()
+ int getConfigTableSize()
+ int getConfigTableInByteArraySize()
+ List<ConfiguringRuleEntrySdiot> getConfigTable()
+ List<ForwardingRuleEntrySdiot> getForwardTable()
+ ArrayList<ConfigRuleEntry> getConfigTableInByteArray()
+ ArrayList<ForwardRuleEntry> getForwardTableInByteArray()
+ List<AttachedSensor> getAttachedSensorList()
+ Pair<Level, String> getLogToBePrinted()
+ NodeAddress getMyAddress()
+ int getNet()
+ NetworkPacketSdiot getNetworkPacketToBeSend()
+ void rxRadioPacket(NetworkPacketSdiot np, int rssi)
+ void startSdvs()
+ void startNode()
+ void timer()
+ void timerSDVS()
- boolean compare(int op, int item1, int item2)
- FunctionInterfaceSdiot createServiceInterface(byte[] classFile)
- int doOperation(int op, int item1, int item2)
- int getOperand(NetworkPacketSdiot packet, int size, int location, int value)
+ int getOperandSdiot(NetworkPacketSdiot packet, String value)
- void initForwardRuleTable()
- void initForwardRuleTableInByteArray()
+ void addToForwardRuleTable(ForwardingRuleEntrySdiot fwRule)
+ void addToConfigRuleTable(ConfiguringRuleEntrySdiot newRule)
+ void addToForwardRuleTableInByteArray(ForwardRuleEntry fwRule)
+ void addToConfigRuleTableInByteArray(ConfigRuleEntry newRule)
- void initConfigRuleTable()
+ void initConfigTableInByteArray()
+ void exeConfigTable()
+ void exeConfigTableInByteArray()
+ List<ConfiguringRuleEntrySdiot> getNonExeRules(List<ConfiguringRuleEntrySdiot> configTable)
+ void exeConfigRules(List<ConfiguringRuleEntrySdiot> nonExecutedRules)
+ void exeConfigRulesInByteArray(List<ConfigRuleEntry> nonExecutedRules)
+ List<ConfigRuleEntry> getNonExeRulesInByteArray(List<ConfigRuleEntry> configTableInByteArray)
+ void showConfigTable(List<ConfiguringRuleEntrySdiot> configTable)
+ void showConfigTableInByteArray(List<ConfigRuleEntry> configTable)
+ void showForwardTable(List<ForwardingRuleEntrySdiot> forwardTable)
+ void showForwardTableInByteArray(List<ForwardRuleEntry> forwardTableInByteArray)
+ void doGetAction(ConfiguringRuleEntrySdiot cfRule, NodeAddress reqSdvsAddr)
+ void doSetOnAction(ConfiguringRuleEntrySdiot cfRule, NodeAddress reqSdvsAddr, List<AttachedSensor> attachedSensors)
+ void doSetOffAction(ConfiguringRuleEntrySdiot cfRule, NodeAddress reqSdvsAddr, List<AttachedSensor> attachedSensors)
+ void doGetActionInByteArray(ConfigRuleEntry cfRule, NodeAddress reqSdvsAddr)
+ double getTimeInMinutes(double timeInMiliSeconds)

```

```

+void doSetOnActionInByteArr(ConfigRuleEntry cfRule, NodeAddress reqSdvsAddr, List<AttachedSensor> attachedSensors)
+void doSetOffActionInByteArr(ConfigRuleEntry cfRule, NodeAddress reqSdvsAddr, List<AttachedSensor> attachedSensors)
+int getCountDataPck()
+void updateConfTtl(ConfiguringRuleEntrySdiot rule)
+int countGETStatus(String reqSID)
+int countSIDONStatus(String reqSID)
+int countSIDOFFStatus(String reqSID)
+int countGETStatusForByteArr(String reqSID)
+int countSIDONStatusForByteArr(String reqSID)
+int countSIDOFFStatusForByteArr(String reqSID)
-void initStateRegister()
-boolean matchRule(FlowTableEntrySdiot rule, NetworkPacketSdiot packet)
-boolean matchFwdRule(ForwardingRuleEntrySdiot rule, NetworkPacketSdiot packet)
-boolean matchWindow(WindowSdiot w, NetworkPacketSdiot packet)
-boolean matchFwdWindow(MatchingWindowSdiot w, NetworkPacketSdiot packet)
-BeaconPacketSdiot prepareBeacon()
-BeaconPacketSdiot prepareBeaconSdiot()
-ReportPacketSdiot prepareReport()
-ReportPacketSdiot prepareReportSdiot()
-void runAction(AbstractActionSdiot act, NetworkPacketSdiot np)
-void runActionSdiot(String act, NetworkPacketSdiot np)
-int searchRule(FlowTableEntrySdiot rule)
-int searchConfigRule(ConfigRuleEntry cfRule)
-void updateForwardTable()
-void updateConfigTable()
+void updateConfigTableInByteArr()
#void controllerTX(NetworkPacketSdiot pck)
#void dataCallback(DataPacketSdiot packet)
#void setActive(boolean active)
#NodeAddress getActualSdvsAddress()
#NodeAddress getNextHopVsSdvs()
#NodeAddress getNextHopVsSdvsSdiot()
#int getSdvsDistance()
#void setSdvsDistance(int distance)
#int getSdvsRssi()
#void setSdvsRssi(int rssi)
#void initSdiot()
#void initSdiotSpecific()
#void insertRule(FlowTableEntrySdiot rule)
#void insertConfigRule(ConfigRuleEntry cfRule)
-boolean isAcceptedIidAddress(NodeAddress address)
-boolean isAcceptedIidPacket(NetworkPacketSdiot packet)
#void log(Level level, String logMessage)
-void execWriteConfigPacket(ConfigPacketSdiot packet)
-boolean execReadConfigPacket(ConfigPacketSdiot packet)
#boolean execConfigPacket(ConfigPacketSdiot packet)
#boolean execModifyPacketSdiot(ModifyConfigRulePacketSdiot packet)

```

```

#void execConfigPacketByNode(ConfigPacketSdiot packet)
#void radioTX(NetworkPacketSdiot np)
#void reset()
#void runFlowMatch(NetworkPacketSdiot packet)
#void runForwardingMatch(NetworkPacketSdiot packet)
#void rxBeacon(BeaconPacketSdiot bp, int rssi)
#void rxConfig(ConfigPacketSdiot packet)
#void rxConfigNode(ConfigPacketSdiot packet)
#void rxData(DataPacketSdiot packet)
#void rxDataSdiot(DataPacketSdiot packet)
#void rxHandler(NetworkPacketSdiot packet, int rssi)
#void rxHandlerSdiot(NetworkPacketSdiot packet, int rssi)
#void rxOpenPath(OpenPathPacketSdiot packet)
#void rxReport(ReportPacketSdiot packet)
#void rxRequest(RequestPacketSdiot packet)
#void rxResponse(ResponsePacketSdiot packet)
#void rxSetFwdRuleSdiot(SetForwardRulePacketSdiot packet)
#void rxSetConfRuleSdiot(SetConfigRulePacketSdiot packet)
#void rxModifyConfRule(ModifyConfigRulePacketSdiot packet)
#void rxReqFwdStats(RequestForwardStatsPacketSdiot pck)
#void rxReqConfStats(RequestConfigStatsPacketSdiot pck)
#void rxReqFeatures(NetworkPacketSdiot pck)
#void rxHello(NetworkPacketSdiot pck)
+List<NodeAddress> getAcceptedId()
+List<FlowTableEntrySdiot> getFlowTable()
+ArrayBlockingQueue<NetworkPacketSdiot> getFtQueue()
+HashMap<Integer, FunctionInterfaceSdiot> getFunctions()
+Set<NeighborSdvs> getNeighborTable()
+int getRssiMin()
+int getRuleTtl()
+HashMap<String, Object> getSensors()
+ArrayBlockingQueue<NetworkPacketSdiot> getTxQueue()
+ArrayList<Integer> getStatusRegister()
+List<AttachedSensor> getAttachedSensorlist()
+void setAttachedSensorStatus(String reqSID, String reqStatus, List<AttachedSensor> attachedSensorList)
+String getAnAttachedSensorStatusBySID(String reqSID, List<AttachedSensor> attachedSensorList)
+int countStateOfASdvs()
+int countStateOfASdvsForByteArr()
+int getCounterBySid(String strSID)
+int getCounterBySidForByteArr(String strSID)
+byte[] getPayloadBySID(String SID)
+void sendResultToMininetHost(String reqData, String dst)

```

Figure 4.9 Abstract methods defining fundamental functions of an SDVS and its represented entity

4.6 Use Case Scenario and Practical Implementation

❖ Use case scenario

For the practical realization of the proposed SDVS, we develop a use case scenario in which SDVSs represent physical IoT devices to provide IoT services on demand. In the scenario, there are a number of IoT devices along a street. Sensing readings from the devices may be used for providing traffic load along the street, weather conditions of an area, or to adjust a traffic light according to traffic load over the street. IoT devices are under the control and management of a controller. Each IoT device is attached with sensors, for example, light, humidity, proximity, and temperature. Sensing readings and actuators from the device can be achieved and executed on demand, respectively.

❖ Implementation set up

To demonstrate the above scenario, we implement a Software-Defined Internet of Things (SD-IoT) system that is detailed in Chapter 6. The system represents a cluster of IoT devices. It is composed of three main components (as shown in **Figure 4.10**): i) a software-defined Internet of Things Device (SD-IoTD) controller, ii) a number of SDVSs and their represented sensor tags, and iii) an S-MANAGE protocol that is a communication protocol between the controller and SDVSs. The SDVSs are representations of physical IoT devices. The SD-IoTD controller controls and manages the SDVSs using the S-MANAGE protocol. Details of the design and implementation of the S-MANAGE and SD-IoTD model can be found in Chapter 5 and Chapter 6, respectively. All components of the model are written in Java by using NetBeans 8.2.

The implementation prototype is displayed in **Figure 4.10**. Six real sensor tags have been used to collect real sensing data. Each sensor tag is a TI CC2650STK that communicate with the SDVS via a Low-Bluetooth protocol. Each of them is attached to ten different types of sensors, including light, digital microphone, magnetic sensor, humidity, pressure, accelerometer, gyroscope, magnetometer, object temperature, ambient temperature [114]. We deploy an SD-IoT system on a Raspberry Pi 3 running Raspbian GNU/Linux 9.11 (Stretch). The raspberry is responsible for controlling and managing the sensor tags in the provision of IoT data on demand.

In this implementation, we make use of only five sensor types containing Ambient_Temperature, Light, Pressure, Humidity, and Infrared_Temperature.

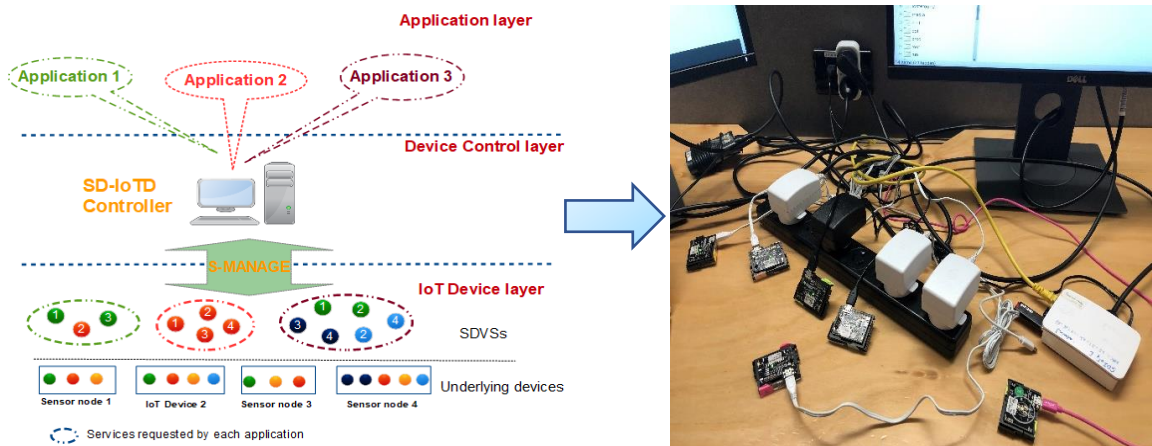


Figure 4.10 Implementation prototype

4.7 Performance Evaluation

This section presents results regarding an SDVS's programmability and efficiency in provisioning IoT services on demand.

4.7.1 SDVS – Feasibility and Programmability

A software driver is installed on the SDVS that represents sensors attached to the sensor tag. Thus, the SDVS can automatically communicate with the sensor tags and achieve sensing values from them when the sensor tags connect to the SD-IoT system via the BLE protocol. As presented in **Figure 4.11b**, the SDVS represents one or multiple sensor nodes (sensor tags) to provide real sensing data for the provision of IoT services on demand. In this implementation, the SDVS represents five sensor types, such as Light, Ambient_Temperature, Pressure, Humidity, and Infrared_Temperature. An example of data collected by the sensor tag is shown in **Figure 4.12**. The log file records a sensor tag's collected data in terms of collecting time, and five sensing data

types such as infrared temperature (ir_temp), ambient temperature (amb_temp), light, pressure, and humidity. The SDVS can perform more actions, such as processing the raw data or transferring them to the upper level.

a) IoT requests handled by the SDVS

SDVS Address:

Act_Type	Req_Servi...	LOC_ID	Freq(s)	Period(m)	Dst	Req_ID	StartTim...	RunTime(...)	TTL(s)	Counter	Executed
SET_OFF	SID01	LOC01	10	5	121	1	1.57517...	32.022	270.0	1	Y
SET_OFF	SID05	LOC01	10	5	121	1	1.57517...	32.018	270.0	1	Y
SET_OFF	SID02	LOC01	10	5	121	2	1.57517...	2.001	300.0	1	Y
SET_OFF	SID03	LOC01	10	5	121	2	1.57517...	2.0	300.0	1	Y

b) Status of represented sensors

SDVS Address:

Service ID	Service Name	Status
SID01	LIGHT	0
SID02	AMBIENT_TEMPERA...	0
SID03	PRESSURE	0
SID04	HUMIDITY	1
SID05	INFRARED_TEMPER...	0

Figure 4.11 SDVS’s programmable features

```

log_0.txt - Notepad
File Edit Format View Help
time_stamp,ir_temp,amb_temp,light,presure, humidity
2019-08-16 13:45:41,18.34375,29.03125,262.72,1004.48,40.4846191406
2019-08-16 13:45:43,21.59375,29.0625,345.44,1004.49,40.4846191406
2019-08-16 13:45:44,17.65625,29.09375,28.71,1004.49,40.4846191406
2019-08-16 13:45:45,16.71875,29.09375,8.29,1004.53,40.4052734375
2019-08-16 13:45:46,19.75,29.125,7.24,1004.5,40.3076171875
2019-08-16 13:45:48,21.09375,29.15625,7.41,1004.54,40.3076171875
2019-08-16 13:45:49,22.4375,29.1875,6.44,1004.55,40.3076171875
2019-08-16 13:45:50,22.25,29.1875,6.52,1004.49,40.2099609375
2019-08-16 13:45:51,22.4375,29.21875,6.6,1004.43,40.2099609375
2019-08-16 13:45:53,22.1875,29.25,17.14,1004.46,40.2099609375
2019-08-16 13:45:54,20.21875,29.28125,16.66,1004.51,40.2099609375
2019-08-16 13:45:55,19.15625,29.3125,12.24,1004.46,40.3076171875
2019-08-16 13:45:56,24.34375,29.3125,11.83,1004.42,40.3076171875
2019-08-16 13:45:58,23.71875,29.34375,12.8,1004.45,40.4052734375
2019-08-16 13:45:59,23.8125,29.375,11.67,1004.46,40.6005859375
2019-08-16 13:46:00,23.25,29.40625,18.67,1004.52,40.7958984375
2019-08-16 13:46:01,9.125,29.4375,11.83,1004.51,40.8935546875
2019-08-16 13:46:03,21.5,29.46875,15.45,1004.43,40.8935546875
2019-08-16 13:46:04,22.21875,29.5,13.19,1004.54,40.8142089844
2019-08-16 13:46:05,19.03125,29.5,48.98,1004.51,40.8142089844
2019-08-16 13:46:06,12.6875,29.5,29.11,1004.54,40.2282714844
2019-08-16 13:46:07,18.0625,29.53125,30.89,1004.47,39.9353027344

```

Figure 4.12 Sensor tag’s log file

Handling multiple IoT requests at a time: As presented in **Figure 4.11a**, the SDVS23 currently handles two IoT requests that have Req_ID “1” and “2”. Each request handled by the SDVS is identified by a request ID (Req_ID). The total number of Req_ID represents the total requests that are served by the SDVS.

Configuring represented sensors in accordance with IoT demands: As displayed in **Figure 4.11b**, the SDVS23 currently represents five sensor types. It can configure the represented sensors to achieve required services such as sensing and actuating. For example, the sensor service SID01, SID02, SID03, and SID05 are configured to be OFF (Status is “0”), while SID04 is ON (Status is “1”).

Configuring and updating the status of represented sensors according to required parameters associated with IoT requests: As presented in **Figure 4.11a**, the SDVS allows the IoT requests to specify metrics related to required services. Take the Req_ID “1” for example, the request requires service SID05 in location LOC01, for 5 minutes. Achieved results need to be sent to the desired destination (as Dst “121”) every 10s. The SDVS also records the time (StartTime) when it starts handling (Executed is “Y”) the request, the remaining time for handling the request (TTL). Moreover, it counts the number of requests interested in a specific service (Counter value).

Updating the SD-IoTD controller: The SDVS can update the controller about changes in its environment and status of its represented sensors, so the controller can resolve conflicts among IoT requests. With such recorded information, the controller can muster availability of underlying resources, and hence orchestrate them for incoming requests.

Collecting real data from represented sensors: For example, a service SID04 is required by an application. SDVS21 is orchestrated to obtain the SID04. As shown in **Figure 4.13**, SDVS21 is configured to achieve the required service with specific requirements.

a) Configuration on SDVS21

Act_Type	Req_Servi...	LOC_ID	Freq(s)	Period(m)	Dst	Req_ID	StartTim...	RunTime...	TTL(s)	Counter	Executed
GET	SID04	LOC01	20	10	121	3	1.57517...	28.011	600.0	1	Y

b) SDVS21's service status

Service ID	Service Name	Status
SID01	LIGHT	0
SID02	AMBIENT_TEMPERA...	0
SID03	PRESSURE	0
SID04	HUMIDITY	1
SID05	INFRARED_TEMPER...	0

Figure 4.13 Configuration status of SDVS21

4.7.2 SDVS – Efficiency

The following result demonstrates the efficient utilization of the proposed SDVS in provisioning IoT services on demand.

Various IoT requests have been sent to the SD-IoTD system. These requests may come to the system at the same time or separately. We try to send a number of requests simultaneously to the SD-IoTD system, for example, 10, 30, 50, 70, 90 requests. Each request may demand one, two, three, four, or five services.

Figure 4.14 shows the efficiency of the SDVS in response to an increasing number of sensor service requests. Even there is a significant rise in the volume of simultaneous incoming requests, the response time of the SDVS experiences a light growth that is much less than the increase in the number of input requests. For example, to respond to 10 concurrent requests, regardless of the number of required services per request, an SDVS needs about 379.3ms on average. When the number of concurrent arriving requests increases by nine times, the response time of the SDVS only rises about 1.8 times. In addition, when the number of services required per request grows five times, the corresponding response time of the SDVS to each request rises about 1.5 times. For instance, when the number of input requests is 10, to response to each request for 1,2,3,4, and 5 services, the SDVS needs about 23, 34, 39, 44 and 48ms, respectively. This pattern happen similarly to the case when the number of input request is 30, 50, 70, and 90.



Figure 4.14 SDVS’s average response time for one request per multiple requests

4.8 Summary

In this chapter, we introduce a software-defined virtual sensor (SDVS) with new concepts to reshape the SDN and NFV technologies and support the provision of IoT services on demand. SDVS is designed to enable IoT devices to be programmable on demand in response to IoT application requests. A detailed design is provided. The implementation results demonstrate the feasibility and efficiency of the proposed SDVS in the provision of IoT applications on demand. In Chapter 7, we will investigate the integration of the SDVS into a large-scale software-defined IoT infrastructure.

Chapter 5 S-MANAGE Protocol

5.1 Introduction

In Chapter 4, we proposed the software-defined virtual sensor (SDVS) as a representation of sensors/IoT devices in the provision of IoT services on demand. The SDVS allows constrained IoT devices to be programmed according to the SDN and NFV principles. To control and manage the SDVS in accordance with the SDN principles, it requires an efficient but light-weight protocol that suits the limitations and features of sensor/IoT devices. However, the SDN Southbound Interface (SBI) protocol was designed to handle SDN networking 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 networking devices, and this introduces complexity to already constrained IoT devices.

This chapter proposes an S-MANAGE protocol that preserves the SDN-NFV paradigm but provides a solution for controlling and managing IoT resources in the provision of IoT services on demand. The S-MANAGE is designed both to configure SDVSs and to control the behavior of the underlying networked IoT resources. This chapter investigates the design and practical implementation of the S-MANAGE protocol. This work has been published in [19]. The function of the S-MANAGE protocol in the overall large-scale software-defined Internet of Things (LSSD-IoT) model is depicted in **Figure 5.1**.

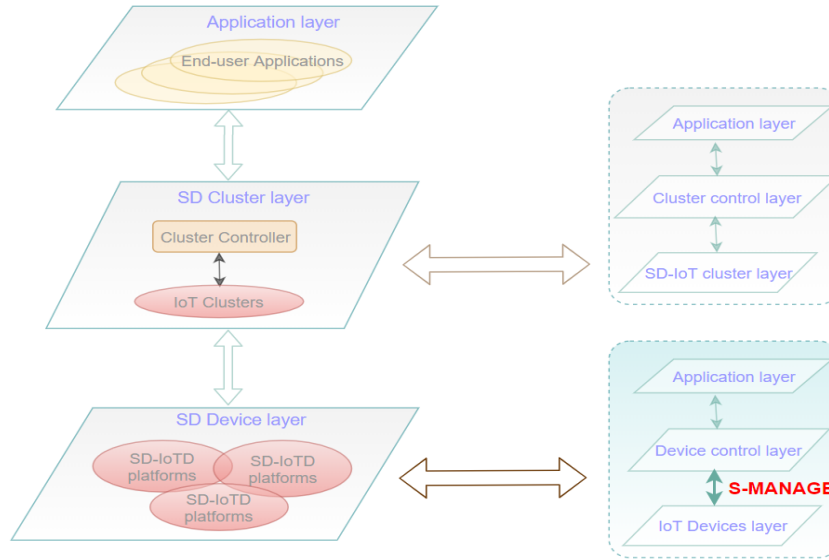


Figure 5.1 S-MANAGE protocol in relation to the LSSD-IoT model

The remainder of this chapter is organized as follows. Section 5.2 describes the functions of the proposed S-MANAGE protocol. Section 5.3 gives detailed specifications of the S-MANAGE protocol. Section 5.4 presents the software implementation of the protocol. Section 5.5 demonstrates the implementation prototype and performance evaluation. Section 5.6 concludes this chapter.

5.2 S-MANAGE in Relation to SD-IoT Model

Before investigating details of the proposed S-MANAGE protocol, we introduce a context where the S-MANAGE protocol fits in. The S-MANAGE is designed to enable the programmability of not only the functionality but also the networking behavior of IoT devices. To realize the proposed features of the S-MANAGE protocol, we describe it in the context of a software-defined Internet of Things (SD-IoT) model. The model is proposed in Chapter 6.

With the proposed SD-IoT model, we achieve three main aims: i) controlling and managing IoT infrastructures according to IoT application demands; ii) allowing heterogeneous IoT devices to participate in an SD-IoT system, and iii) extending the scope of the SD-IoT, allowing it to be deployed and share resources in wider SDN domains for global or end-to-end IoT applications.

The proposed SD-IoT model is also structured in three layers, including the application layer, the device control layer, and the IoT device layer, as depicted in **Figure 5.2**.

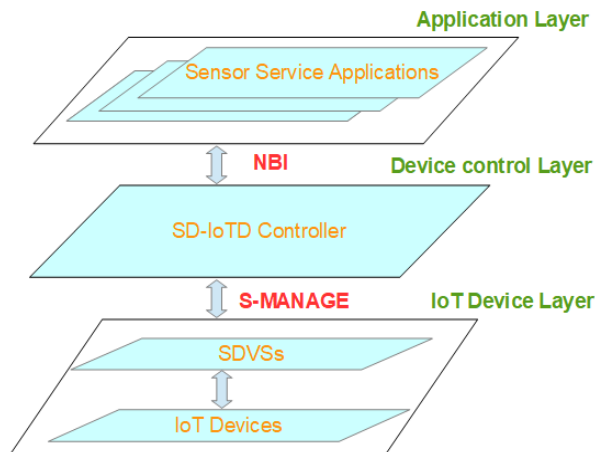


Figure 5.2 SD-IoT model

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

The control layer accommodates a software-defined Internet of Things Device (SD-IoTD) 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 the requirements 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-IoTD controller and the physical devices. The virtual data layer enables the controller to manage and control the underlying resource in the physical data layer.

❖ **SD-IoTD controller**

The SD-IoTD 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-IoTD controller makes it possible for the application layer to specify their 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 application requirements. To configure the underlying resources, the controller makes use of an SBI defining resource-specific messages to configure these devices.

❖ **Software-Defined Virtual Sensor (SDVS)**

The SDVS is defined as representative 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 software-defined function (SDF). The core modules enable the SDVS to behave 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. Specifically, 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.

❖ **S-MANAGE protocol**

The S-MANAGE protocol is an SBI between the SD-IoTD controller and the virtual data layer. Via the SBI, the controller can manage and configure the SDVSs and via which their represented devices are configured.

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 behaviors of SDVSs.

5.3 S-MANAGE Protocol

In traditional IP networks, a router is 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, 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 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, the OpenFlow SDN network still requires OF-CONFIG or other protocols for device configuration. What we need is a protocol that is both light-weight specifically designed for resource-constrained devices 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. The S-MANAGE protocol is proposed as a southbound interface between the SD-IoTD controller and the virtual data layer. Via the southbound interface (SBI), the controller can both manage and configure SDVS in this layer.

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

The protocol is proposed according to the spirit of two protocols, OpenFlow [115] and OF-CONFIG [116]. The OpenFlow focuses on flow rules as setting, modification, deletion, or adding

rules for controlling the forwarding behavior of OpenFlow switches. Meanwhile, the OF-CONFIG enables configuring an OpenFlow Switch itself as the setting of the port number, IP address, or interfaces. It should be noted that the proposed S-MANAGE is a new design for both controlling the behavior and programming the configuration of the underlying resource-constrained IoT devices. It simplifies the matching concept from OpenFlow, but it is not an adaptation of OpenFlow with additional features of OF-CONFIG. The adaptation of OpenFlow has not been successful with many previous research attempts, as discussed in Chapter 2.

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 an SDVS on how to handle an arriving packet, while the configuring table guides an SDVS to configure its represented underlying nodes. It should be noted that an SDN switch is considered as a set of flow tables, each with many rows of flow rules, and traffic flow rules are determined by the SDN controller and installed at the SDN switches so that they can match passing traffic flows and forward them according to the rules. As IoT devices are not routing devices, the situation is much simpler, the forwarding table in the SDVS is just an instruction from the SD-IoTD controller to tell the SDVS what to do with the received packet. Much simpler header and only simple header matching operations are required.

The protocol allows the controller to i) install instruction tables on an 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 as the number of processed packets or sensor services required by IoT applications.

In addition, via the protocol, an 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 operation of the S-MANAGE protocol can be clarified via the sequence diagram (**Figure 5.3**). For example, the SD-IoT controller requests forwarding statistics from an SDVS. Both SD-IoT controller and SDVS maintain their connection by exchanging Hello messages. The SD-IoT controller sends a Forwarding Statistics Request message to the SDVS, and the SDVS replies to

the request by sending a Forwarding Statistics Response message back the controller. With the established connection, the SDVS can update the SD-IoT controller on its forwarding statistics.

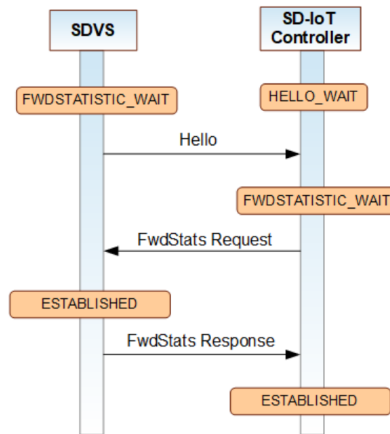


Figure 5.3 Sequence diagram for Forwarding Statistics achievement

The 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 an SDVS is programmed and operates based on these tables’ instructions. Details of the protocol design are described in the following sections.

5.3.1 S-MANAGE Header

The structure of the S-MANAGE packet comprises a header and a payload. All S-MANAGE messages begin with an S-MANAGE header, as depicted in **Figure 5.4**. The header size is 10 bytes. It includes the following parts.

Src.Addr	Dst.Addr	Nxt.Hop	Type	Msg.ID	TTL	Pkt.Lgth
(2bytes)	(2bytes)	(2bytes)	(1byte)	(1byte)	(1byte)	(1byte)

Figure 5.4 S-MANAGE header

- Source Address (Src.Addr) is an address of a source sending a packet.
- Destination Address (Dst.Addr) is a destination address of a packet.

- Next hop address (Nxt.Hop) is an address of a hop in the list providing the path of a packet from the source to the destination.
- Type indicates a packet type. The action on an incoming packet depends on the packet type.
- Packet length (Pkt.Lgth) indicates the length of a packet, including its header and payload.
- TTL is “time to live” of a packet. It is reduced by one at each hop.
- Message-ID (Msg.ID) is an identifier of the packet type.

5.3.2 S-MANAGE Message types

The payload carries the content of a packet. Different types of packets carry different kinds of information which 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 the controller), and iii) symmetric (controller/SDVS to SDVS/controller), as presented in **Figure 5.5**. 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.

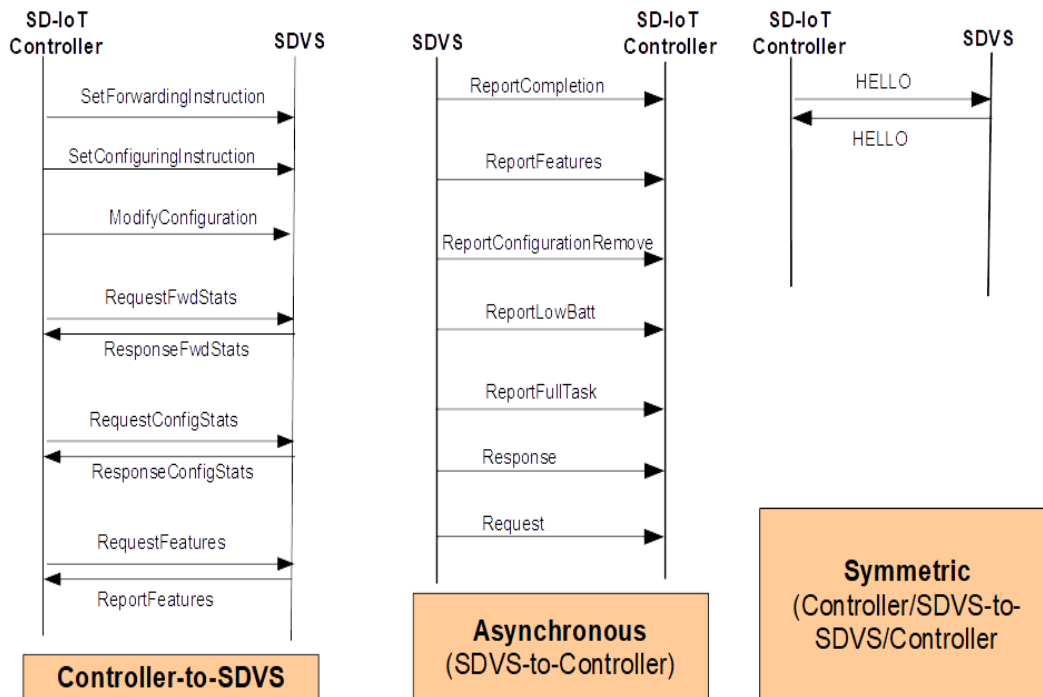


Figure 5.5 S-MANAGE message types

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 the installation of forwarding and configuring instructions on an SDVS need no responses from the SDVS. This category includes messages as SetForwardingInstruction, SetConfiguringInstruction, ModifyConfiguration packets. However, if the controller demands an SDVS’s attributes or status, it needs the SDVS’s responses.

SetForwardingInstruction/SetConfiguringInstruction: Enables the controller to install a forwarding/configuring instruction on an SDVS’s forwarding/configuring table, and to respond to an SDVS’ requests for a forwarding/configuring instruction respectively.

ModifyConfiguration: Modifies a configuration instruction.

RequestFwdStats/RequestConfigStats: Gets statistics of a forwarding or configuring instruction respectively.

ResponseFwdStats/ResponseConfigStats: Is 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.

RequestFeatures/ResponseFeatures: Gets an SDVS's information about its sensor service list, the services' status, or driver of the underlying node.

Asynchronous message type (SDVS-to-Controller): This message type is sent from an SDVS to a controller without any request from the controller. It enables the SDVS to ask for instruction on handling incoming packets and to update the controller on changes of its underlying sensor nodes regarding their active/idle status or completion of a required task.

Report packet: Reports the status and behavior 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 in the considering list of the controller (ReportFullTask).
- e. Inform the controller about the completion of a required service by an SDVS (ReportCompletion).

Response message type: Sends required services back to a required destination.

Request message type: Requests an instruction for its operation. Particularly, if an 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.

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.

5.3.3 Forwarding Table Specifications

The forwarding table contains instruction entries as rows of the table. This table is composed of three main components, matching window, action window, and statistic window, as presented in **Figure 5.6**. The matching window is matched against the header fields of 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 BROADCAST				
	<			FORWARD BROADCAST				
	<=			CONTINUE				

Figure 5.6 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 this 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 a 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*: This 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 as FORWARD_UNICAST, FORWARD_MULTICAST, FORWARD_BROADCAST, DROP, MODIFY, or CONTINUE.
- b) *Value 1*: Different action types result in the different meaning 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. For the FORWARD_UNICAST, FORWARD_MULTICAST, and FORWARD_BROADCAST action type, it demands the unicast, multicast, and broadcast address, respectively.
- c) *Value 2*: A replacement for the old value.

Statistic window: With a focus on the efficient programming of underlying sensors/ IoT devices, statistics 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 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.

5.3.4 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 shown in **Figure 5.7**.

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								
MODIFY								

Figure 5.7 Configuring table structure

Configuring window: The configuring window includes three components: the 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: This is an executing period of 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 entry 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 the 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.

5.4 Software Implementation

This section describes the software implementation of the config table and the forwarding table.

❖ Configuring table in software

Classes needed for establishing the configuring table are displayed in **Figure 5.8**. The classes are for forming a configuring entry in the configuring table.

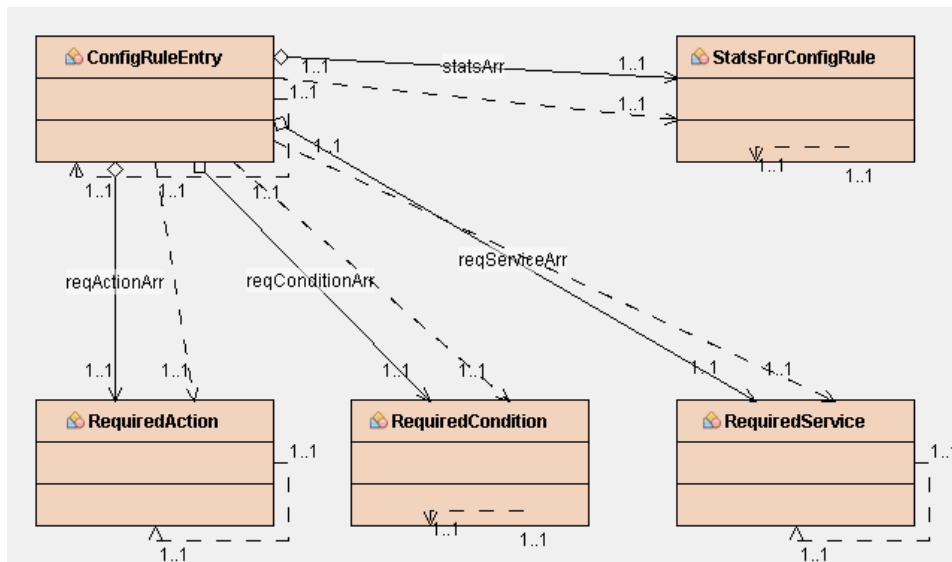


Figure 5.8 Class diagram of the configuring table

The `configRuleEntry` class is shown in **Figure 5.9**. The class provides methods for assessing or configuring a configuring entry in the configuring table.



Figure 5.9 Details of a configuring entry

❖ **Forwarding table in software**

Figure 5.10 presents classes that are used for forming a forwarding entry in the forwarding table.

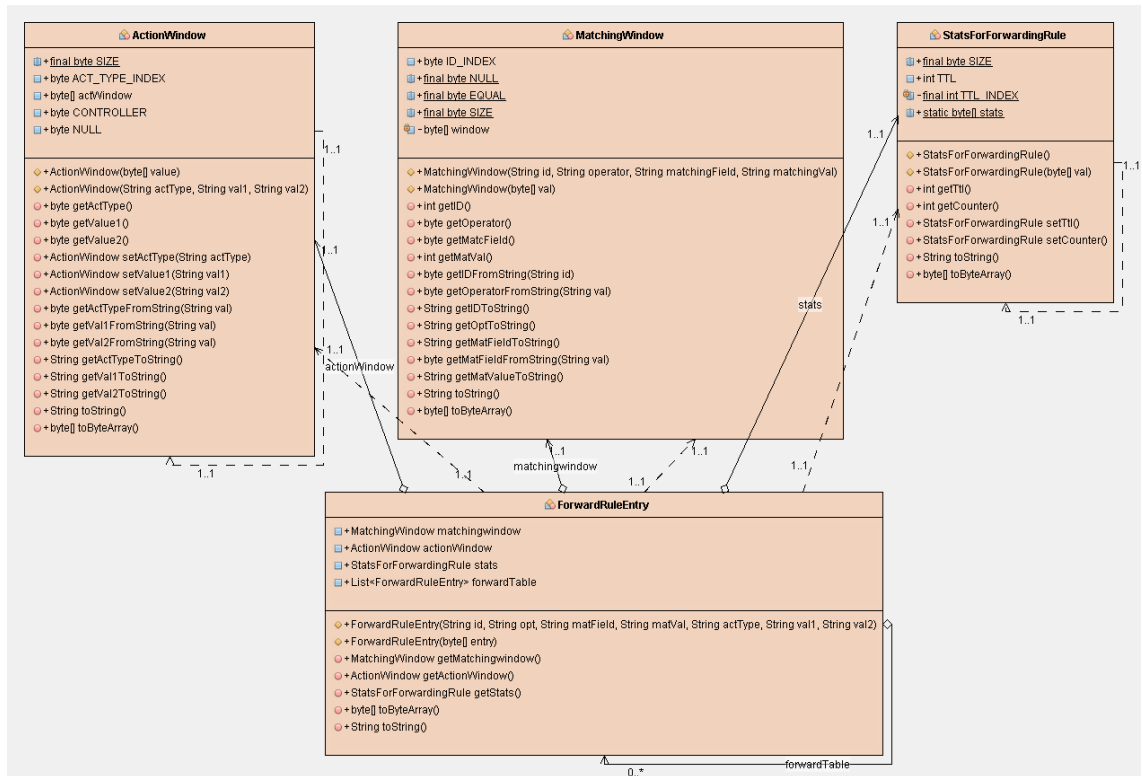


Figure 5.10 Class diagram of the forwarding table

❖ S-MANAGE packets

S-MANAGE packet (Figure 5.11) is developed from a base packet that is a network packet.

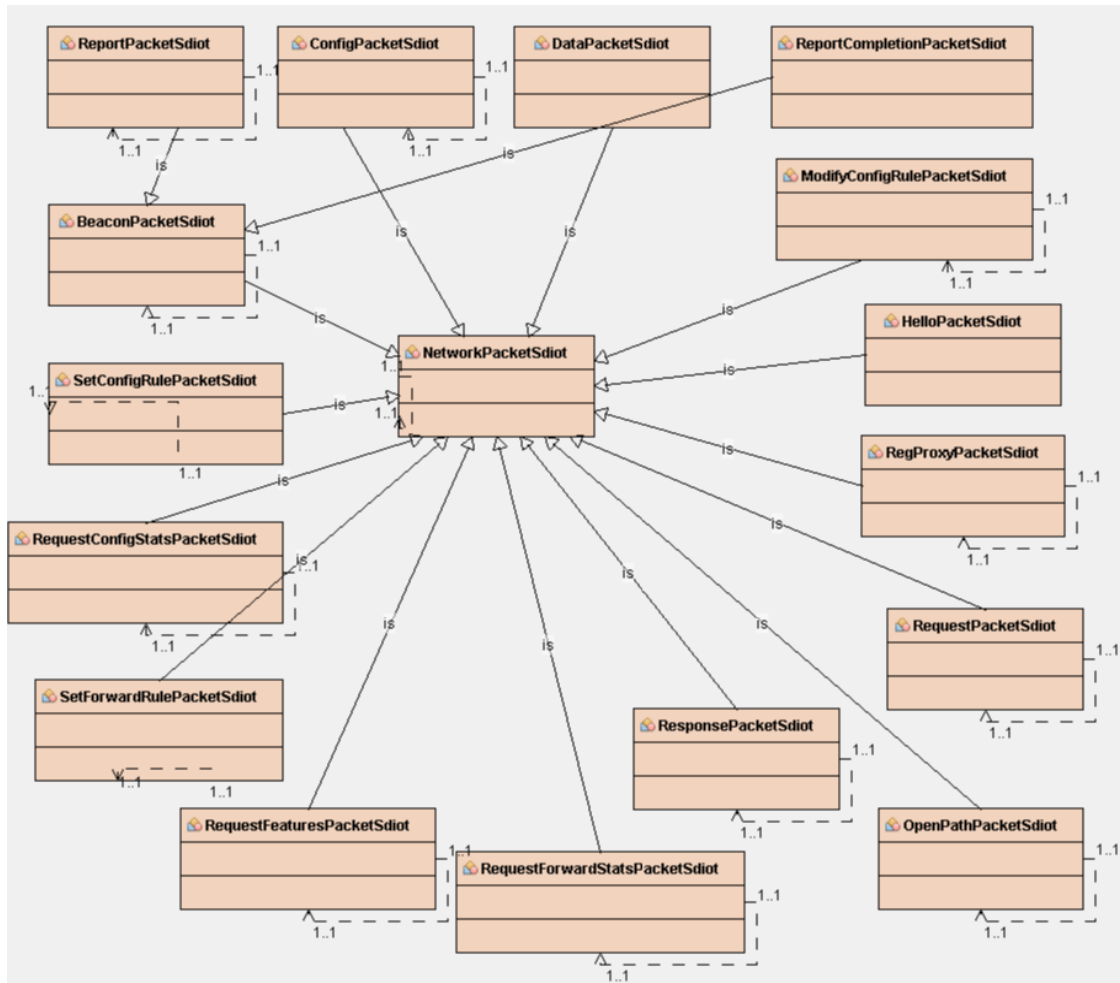


Figure 5.11 Class diagram of S-MANAGE packets

5.5 Implementation and Performance Evaluation

5.5.1 Implementation Set up

To demonstrate the performance of the proposed S-MANAGE protocol, we develop an SD-IoT environment where the S-MANAGE is utilized as a control and management protocol. The controller configures SDVSs using the S-MANAGE.

The implementation is developed from our preliminary implementation [23] and the Java-based open-source platform [117] into a comprehensive design and implementation. The SD-IoT

model is a software platform written in Java by utilizing Netbeans 8.2. It is connected to data storage that is built using MySQL. The three main elements comprise the SD-IoTD controller, the S-MANAGE protocol, and the SDVS. The implementation prototype is depicted in **Figure 5.12**.

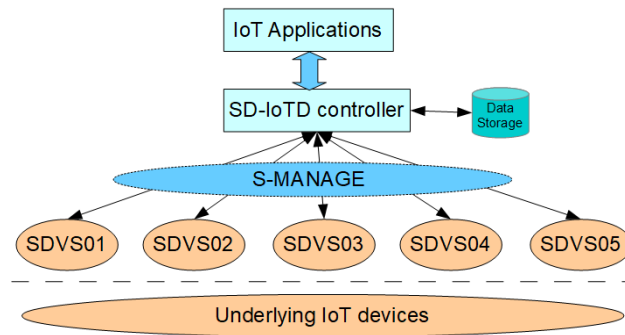


Figure 5.12 Implementation prototype

Three software modules, including control, southbound interface, and virtual representation, are responsible for the SD-IoTD 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 tables, and configuring tables. The virtual representation module contains Java classes for defining the core of an 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.

5.5.2 Performance Evaluation

Users send requests to the controller via a GUI interface. The request is a set of parameters, including Act_Type, SIDs, Freq, Per, Dst., LOCID, ReqID. The request parameters indicate the required services (SIDs), 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.

The following figures illustrate a forwarding table, a configuring table, and a sensor service list of an SDVS in the network of five 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.

Figure 5.13, Figure 5.15, and Figure 5.17 demonstrate the results of the first case. They present the status of the SDVS02 before it is programmed by the controller, such as its forwarding instructions (**Figure 5.13**) and configuring instructions (**Figure 5.15**), and sensor services status (**Figure 5.17**).

Figure 5.14, Figure 5.16, and Figure 5.18 reveal the results of the second case. When there is a request for sensor services, the SDVS02 is orchestrated by the controller to handle the request. The controller deploys associated instructions to the SDVS’s forwarding table (**Figure 5.14**), configuring table (**Figure 5.16**). Thus, its sensor services are configured, and their status is changed accordingly (**Figure 5.18**).

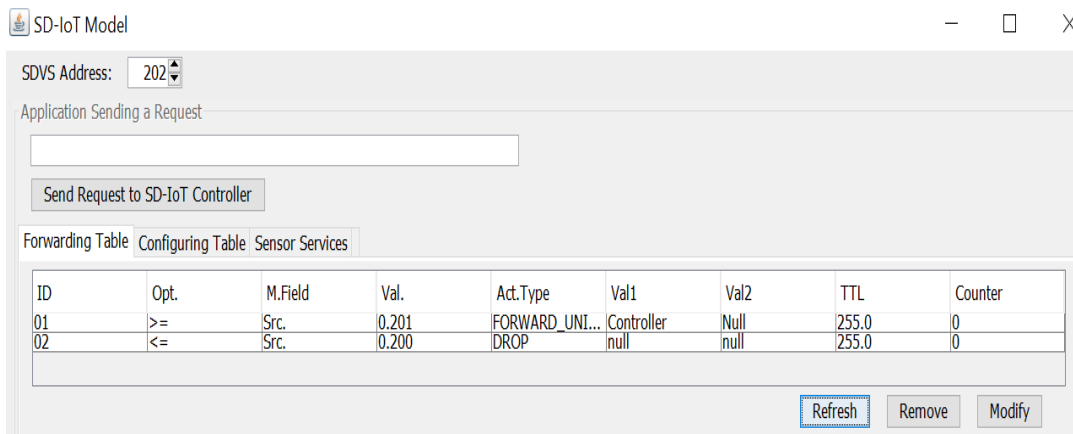


Figure 5.13 Forwarding table status before configuration

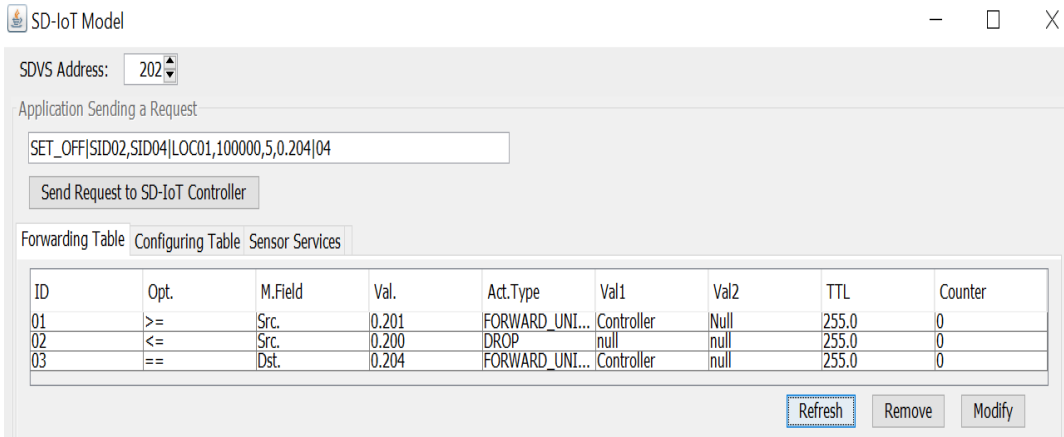


Figure 5.14 Forwarding table status after configuration

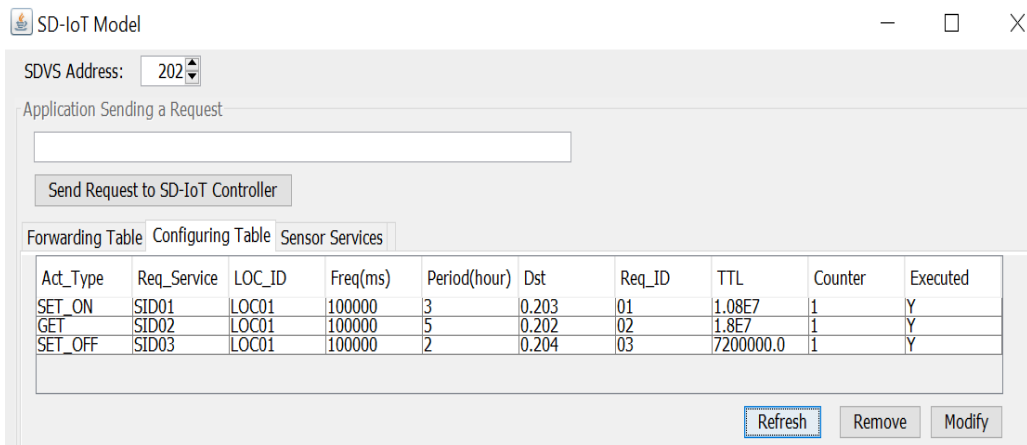


Figure 5.15 Configuring table status before configuration

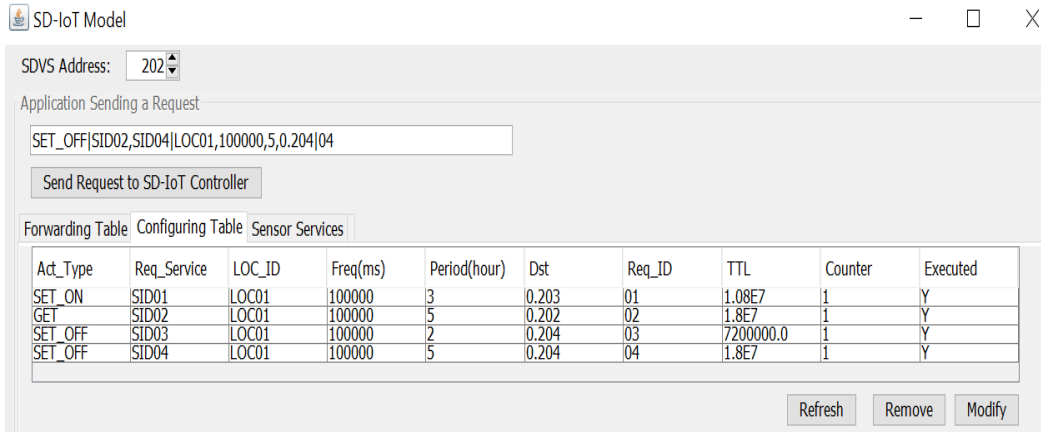


Figure 5.16 Configuring table status after configuration

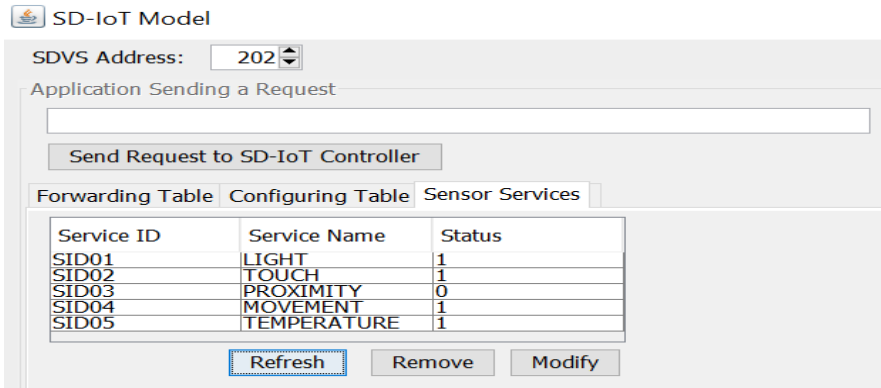


Figure 5.17 Sensor service status before configuration

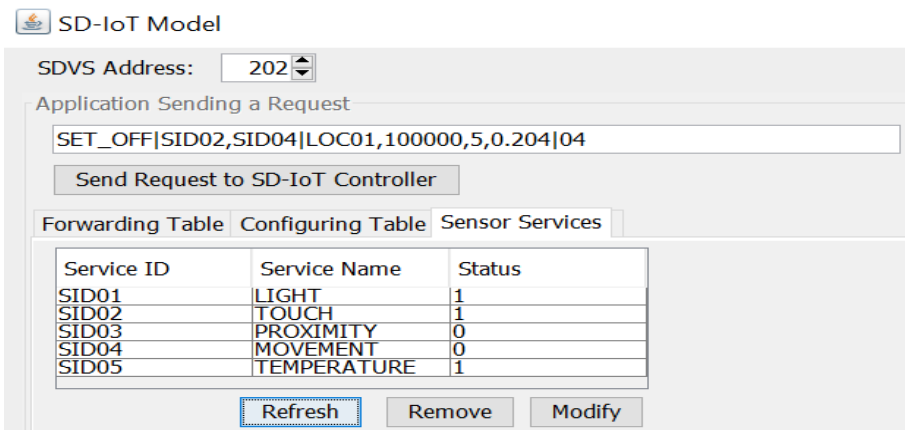


Figure 5.18 Sensor service status after configuration

5.6 Summary

In this chapter, we proposed a design and implementation of the S-MANAGE protocol to address the challenges of configuring and programming IoT devices in the SD-IoT model for provisioning IoT services on demand. The S-MANAGE was designed based on OpenFlow and OF-CONFIG to IoT devices in both forwarding behaviors and their functionalities. Details of the design are provided. We also proposed an SDVS as an interface enabling the controller to control and manage physical IoT devices via the S-MANAGE protocol. The software implementation of the proposed S-MANAGE was developed and deployed. The implementation performance was presented to demonstrate the feasibility of the proposed protocol. The proposal enables further research and development on interoperability and orchestration of heterogeneous sensors/IoT devices for the provision of diverse IoT services on demand.

Chapter 6 Software-Defined Internet of Things (SD-IoT) Model

6.1 Introduction

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 [89]. The SDN provides solutions to programmability, agility, flexibility, and end-to-end connectivity challenges, which are associated with the management of real-time traffic flows and dynamic traffic patterns [118]. The SDN approach addresses many existing problems concerning network management and provisioning of resources required by network services. It can change the functionality of physical networks as well as devices in real-time to meet the requirements of IoT applications [119].

However, challenges remain when applying the SDN paradigm to the constrained WSN/IoT [64]. 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 issues concerning IoT platform architectures and the provision of IoT services on demand, there are remaining challenges, such as scalable and dynamic resource discovery and composition, context-awareness, integration of intelligence, interoperability, reliability, security, privacy, and system-wide scalability [120].

In this chapter, we propose a software-defined Internet of Things (SD-IoT) model. The model enables the programmability of underlying devices with an IoT cluster by leveraging the proposed SDVSs and S-MANAGE protocol, that have been discussed in Chapters 4 and 5, respectively. The provision of IoT services by leveraging the S-MANAGE, SDVS, and the SD-IoT model has been published in [20, 22, 23]. The model is accommodated in the SD device layer of an overall LSSD-IoT model (**Figure 6.1**).

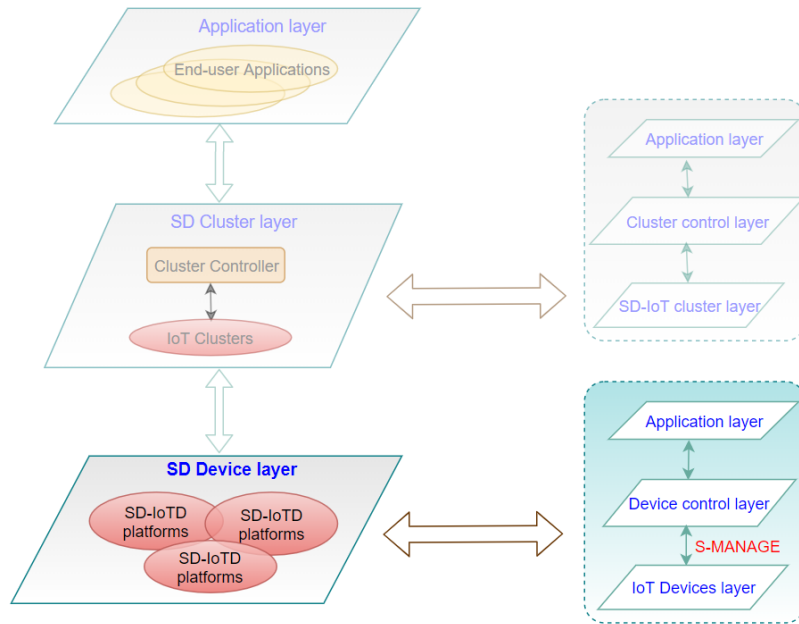


Figure 6.1 SD-IoT model in relation to the LSSD-IoT model

The remainder of this chapter is organized as follows. Section 6.2 describes the overall SD-IoT architecture. Section 6.3 presents the structure and functionalities of the SD-IoTD controller. Section 6.4 describes the software implementation of the SD-IoTD controller. Section 6.5 demonstrates an implementation scenario. Section 6.6 presents the performance evaluation of the SD-IoT model. Section 6.7 concludes this chapter.

6.2 SD-IoT Model

With consideration for the architecture and the application aspects discussed in the earlier section, we propose a Software-defined Internet of Things (SD-IoT) model that embraces the SDN and NFV principles. To reap the benefits of SDN, the SD-IoT model is also structured in three layers: the application, the device control, and the IoT device layers, as shown in **Figure 6.2**.

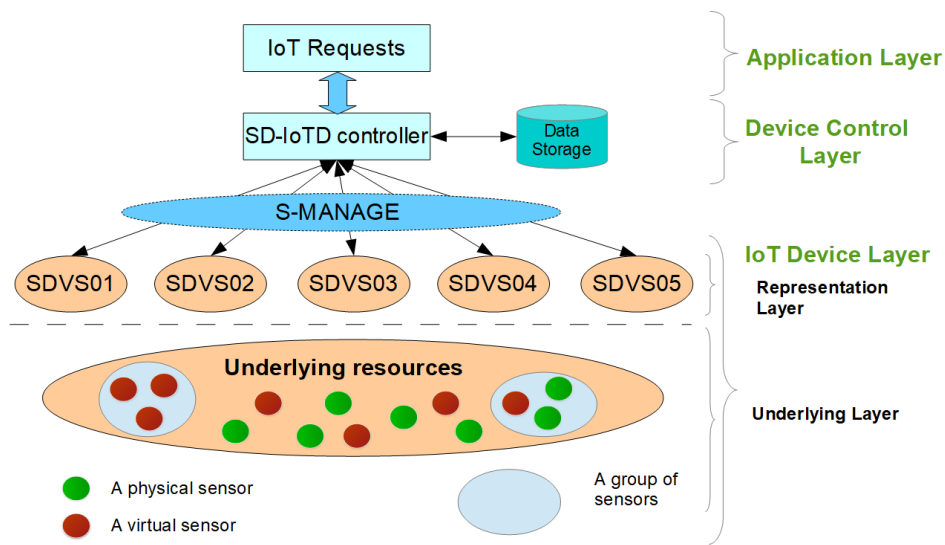


Figure 6.2 SD-IoT model

Application layer: This is where developers can deploy their IoT applications over abstraction of the underlying IoT infrastructure.

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

IoT device layer: This layer hosts SD-IoT resources. Different from the SDN data layer, this layer is designed with two sublayers, called representation and underlying layers. The representation layer is an interface between the SD-IoTD controller and the underlying resources. This layer enables the controller to manage and control the underlying resources according to application demands. The underlying resources consist of physical or virtual sensors or a group of physical/virtual sensors.

With the proposed SD-IoT model, we address three main aims: i) enabling autonomous configuration and management of heterogeneous IoT devices in an SD-IoT system, ii) programming services on demand, and iii) enabling large scale IoT applications through modulation of SD-IoT subsystems.

6.2.1 Software-Defined Virtual Sensor (SDVS)

The SDVS is introduced as an interface between the SD-IoTD model and physical sensors/IoT devices. The SDVS is a software entity that functions as a virtual sensor that addresses the limitations of physical sensors/actuators/IoT devices and possesses capabilities for adapting itself in interacting with the surrounding environment and its controller for providing desired services. It provides the SD-IoTD controller with the capability of its represented devices. In addition, it programs its represented devices in accordance with demands from the SD-IoTD controller. This work has been accepted for publication [18]. It is described in Chapter 4.

6.2.2 S-MANAGE Protocol

The prime purpose of the protocol between the SD-IoT controller and SDVS is for controlling and managing functioning and forwarding behaviour of IoT devices in provisioning IoT services on demand. Current SDN-based solutions mainly put effort on particular challenges of the integration of the SDN principles to WSN/IoT environment and fail to fully address requirements for the integration. Regarding the southbound interface, as discussed in section 2.7, several efforts have attempted to modify, extend or adapt the OpenFlow protocol. However, their contributions have been limited to theoretical proposals, or partial consideration for the forwarding aspect of

the sensor nodes rather than looking at autonomous configuration management of the node and its functions. Therefore, the S-MANAGE protocol is proposed as a control and management protocol between the SD-IoTD controller and virtual sensors within the representation layer. It enables the SD-IoTD controller to manage and program the SDVSSs within the representation layer to achieve the required services and deliver them to the right destinations. The proposed S-MANAGE protocol is described in Chapter 5.

6.3 SD-IoTD Controller

6.3.1 SD-IoTD Controller – Functional Components

The SD-IoTD controller is a bridge between an IoT cluster and the LSSD-IoT system. As for an IoT cluster, it is a manager that musters IoT devices capabilities within the cluster and can configure the managed resources in response to IoT demands. Regarding the LSSD-IoT system point of view, the SD-IoTD controller represents its own IoT cluster. It is responsible for reporting on its capability in provisioning IoT services on demand and orchestrating its resources to provide requested services. The basic set of components of the SD-IoTD controller (**Figure 6.3**) may consist of a service handler (SH), an underlying handler (UH), a resource orchestrator (RO), a configuration manager (CM), a resource manager (ReM), a topology manager (TM), a sensor service life cycle monitor (SSLCM), a routing manager (RM), and a data storage (DS). The proposed functions and mechanisms of the SD-IoTD controller has been published in [22], [23].

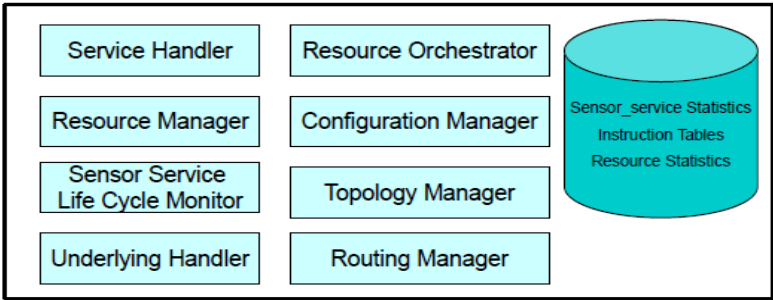


Figure 6.3 SD-IoTD controller structure

Service Handler provides an abstraction of the SD-IoT resources for IoT applications. It interprets application requests in their high-level language and translates them to the controller's language. For instance, it provides application requirements for Resource Orchestrator.

Resource Manager manages SDVSs and through that manages their corresponding underlying resources. It is responsible for updating the resource orchestrator on the statuses of the SDVSs and their underlying resources.

The database provides updated information for the operation of the controller's core modules, for instance, the Resource Manager. The DB can be accessed by all other core components.

Underlying Handler is called S-MANAGE protocol. It provides a communication interface between the controller and the SDVS. It enables the controller to configure SDVS resources by using S-MANAGE messages as well as to obtain update status of the underlying resources.

Resource Orchestrator orchestrates appropriate SDVSs to provision services required by the application.

Topology Manager manages directly the network of the SDVSs in the representation layer and indirectly the real network of underlying resources in the underlying layer. The network status is updated by the resource manager.

Routing Manager computes a forwarding path between a source and destination in accordance with the network topology.

Configuration Manager generates forwarding and configuring instructions in accordance with the results from the routing manager, and the resource orchestrator.

Sensor Service Life-cycle observes the life-cycle of each request-response transaction, each SDVS, and the association between the SDVS and its underlying resources.

6.3.2 SD-IoTD Controller – Operational Mechanism

The important functions of the SD-IoTD controller are to analyse input requests and to muster its capabilities, so it is able to orchestrate and provision its services on demand. To achieve and perform the functionalities, the SD-IoTD has installed the following mechanisms.

❖ Resource Orchestration Approach

The core module of the controller is the resource orchestrator. The efficiency of the network operation is dependent on how intelligent it orchestrates SD-IoT resources. Its main functions are described below.

It orchestrates the most capable SDVSs that can handle the application request. Firstly, on receiving the application requirements from the service handler, it searches for those SDVSs that are capable of handling the request. To balance and minimize the workload of the SD-IoT resources, among the potential candidates, only the most appropriate ones are selected. In addition, the resource orchestrator may also reuse the sensor services temporarily cached in the database to quickly respond to application requests with similar requirements.

We design an algorithm (**Algorithm 6.1**) for associating an appropriate SDVS, which both satisfies application requests and current state of the underlying resources. On receiving a service request, the resource database is checked for available and capable SDVSs, and a list of potential SDVSs is produced. The “state” of an SDVS indicates its busy level. The higher the state number, the busier is the SDVS. A “state” of each SDVS in the list is checked, the one with the lowest value (the least busy) is selected for the service provision. The state of an SDVS is computed according to the total number of tasks it performs and is updated in accordance with the counter statistics from the configuring table.

Algorithm 6.1: Resource Orchestration

Inputs: required parameters as `action_type`, `services`, `locations`, an updated list of application requests and associated SDVSs

Outputs: the list of SDVSs and associated required services for a request

Switch `action_type` **do**

case "GET":

 get a list of services with GET action in the required locations from the updated application request list

if the list is empty **then**

 get pairs of the best SDVS and the required service

```

else
    get a list of possible SDVSs can provide the required services
    get pairs of the best SDVS and the required service
end if
case "SET_ON":
    get pairs of SDVSs and the required services
case "SET_OFF":
    get pairs of SDVSs and the required services

```

❖ Configuration Approach

The controller needs to ensure that there are no configuration conflicts on an SDVS. Thus, in cases where an SDVS encounters a configurational conflict, a priority scheme is used. The configuration instruction with higher priority is executed first to its completion before the one with lower priority.

Moreover, the overhead in configuration messages needs to be taken into account to achieve light SDVS objects. Thus, to reduce exchanged information between the controller and the SD-IoT resources, the resource orchestrator reuses current instructions on instruction tables because applications with similar interests require similar instructions. Thus, the configuration manager modifies associated instructions that can be reused for the new configuration. Furthermore, one forwarding instruction can be applied for a flow of packets, so memory usage is minimized.

❖ Algorithm for Association of service-resource

We propose an algorithm (**Algorithm 6.2**) for associating appropriate SDVS, which both satisfies application request and current state of the underlying resources. A service request is checked if its desired service exists in the `tab_location_sdvs` or not, and a list T1 including corresponding `sdvs_id` and `sid` (sensor service ID) is created (line 1-2). In line 3-6, if the list T1 is not empty, for each row (called record) in list T1, the `sdvs_id` and `sid` of each record are checked in `tab_sdvs_status` to obtain its “state” value concerning its current tasks. Then, the `sdvs_id` and

sid with the least state value are picked to serve the request. The returned results are the sdvs-id associated with the required sid. If the list T1 is empty, the program exits (line 7-10).

Algorithm 6.2: Association Service-Resource

Input: a set of requests RQ with parameters to require sensor services

Output: a set of associated SDVS_IDs and SIDs

```
1: for each request  $r$  in  $RQ$  do
2:   if ( $r.sid$  &  $r.location\_id$ ) exist in database then
3:      $T1=getListSDVS(r)$ ; //Update T1 with SDVS_IDs by location_id and sid
4:   else
5:      $Exit()$ ;
6:   end if
7:   for each  $r.sdvs\_id$  of each  $r.sid$  in  $T1$  do
8:      $PickBestSDVSbySID(r)$ ; //Return proper SDVS_IDs for SIDs
9:   end for
10: end for
```

6.4 SD-IoTD Controller - Software Implementation

This section describes the SD-IoTD controller's software implementation in terms of functional components and data storage. The overall class diagram of the SD-IoTD controller is shown in **Figure 6.4**.

ControllerAnalyseAppRequest class is responsible for analyzing the input request and interpreting it into requirements that are used by the ControllerResourceAllocation class and ResoureManager class for orchestrating resources. ControllerConnectDatabase class provides a connection between the database and all functional components of the SD-IoTD controller. Via the connection, data stored in the database can be retrieved and updated. ControllerGuiSdiot creates the GUI that allows the SD-IoTD controller to present their capabilities and receive IoT demands from its application domain. ThreadExecInputRequestFrFloodlight handles IoT

requests from the SD-IO TC controller. ControllerDijkstraSdiot class computes forwarding paths between SDVSs. The basic functionalities of the SD-IO TD controller is implemented in classes, including, ControllerInterfaceSdiot, ControllerFactorySdiot, and AbstractControllerSdiot. Detailed implementation of the classes is illustrated in **Appendix 1**.

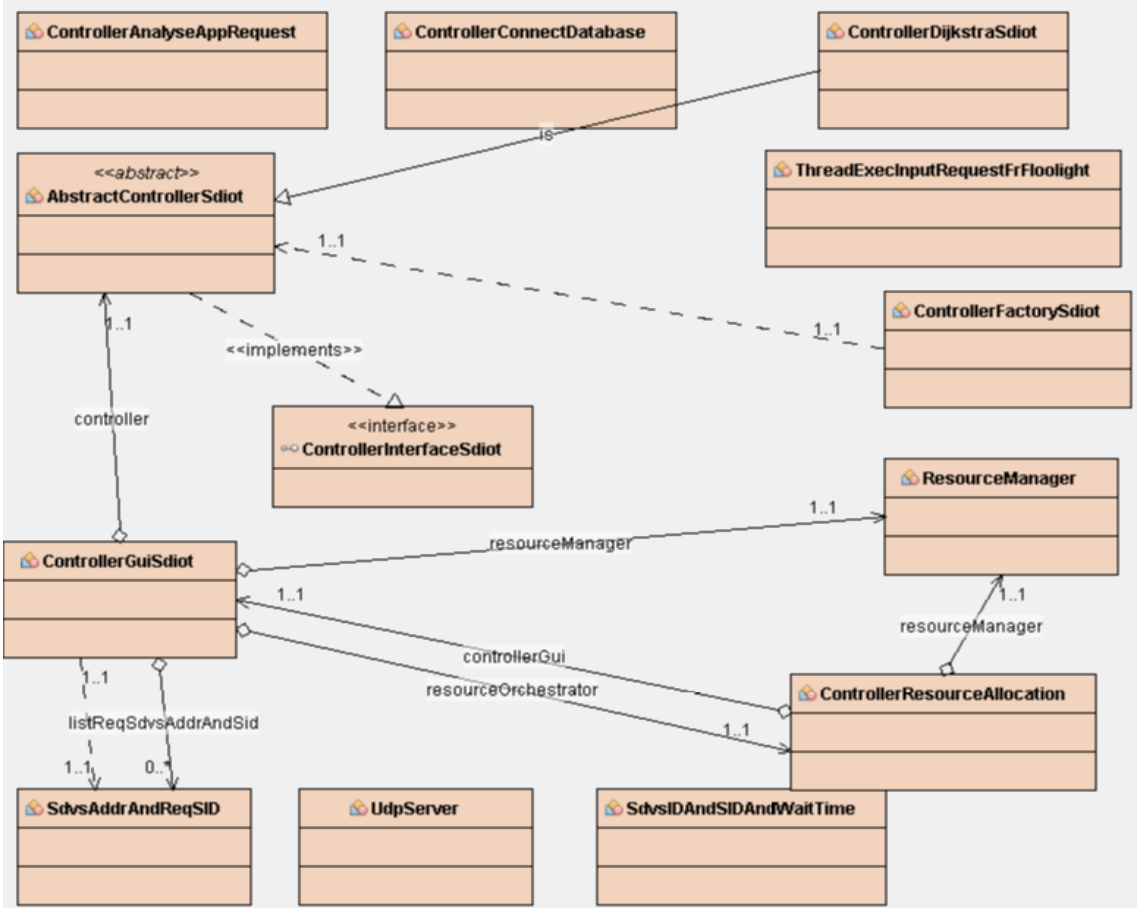


Figure 6.4 Class diagram of SD-IO TD controller

Data Storage: The database is created with MySQL, including three tables (as shown in **Appendix 2**). The tab_location_sdvs table stores SDVS’s information as location_id (location ID of the SDVS, e.g., LOC01, or LOC02), sdvs_id (identification of the SDVS, e.g., SDVS01), and service_id (identification of the sensor service, e.g., SID01). The tab_sdvs_status table includes information as location_id, sdvs_id, service_id and its state which presents the current

task of the SDVS and its related sensor service. The `tab_services` table stores `service_id`, `service_name`, and `service_description` which is presented as user-familiar language, e.g., temperature, humidity, or light. This table provides information for translating user requests.

6.5 SD-IoT Model – Software Implementation

6.5.1 Use Case Scenario

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 6.5**.

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.

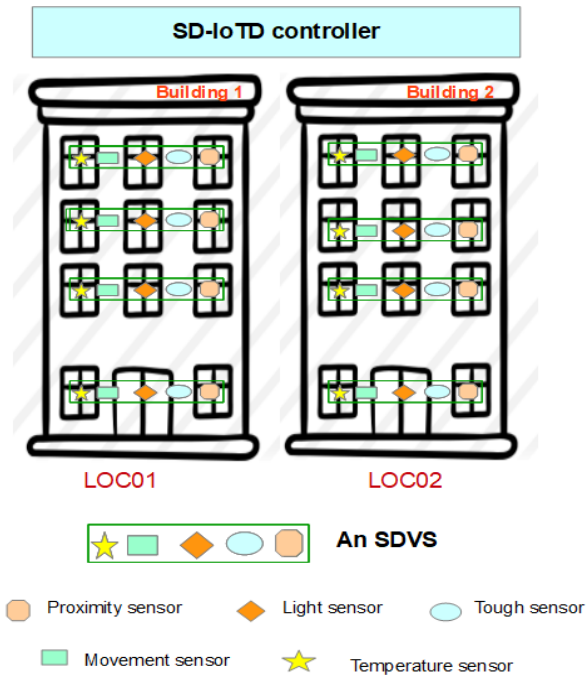


Figure 6.5 Use case scenario

6.5.2 Implementation Scenario

Our aim is to provision IoT services 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 **Figure 6.6**. 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 be 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.

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.

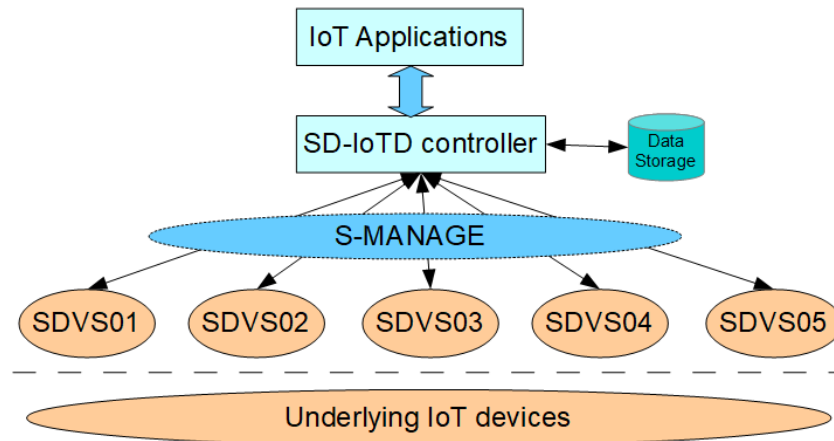


Figure 6.6 Implementation prototype

6.5.3 Implementation Set up

The SD-IoT model is a software platform written in Java and built using Netbeans 8.2 and open-source platform supporting Java dependency classes [117]. It is connected to a database built in MySQL. The three main elements are the SD-IoTD controller, the S-MANAGE protocol, and the SDVS.

Three software modules, including control, southbound interface, and virtual representation, are responsible for the SD-IoTD 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, the forwarding table, the configuring table. The virtual representation module contains classes for defining the core of an SDVS and its software-defined function.

We build a network where the controller communicates with its SDVSs. We establish a database by using 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 the operation of the controller's core modules.

6.6 Performance Evaluation

Implementation results demonstrate the expected features of the proposed S-MANAGE protocol in provisioning IoT services 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 6.7** and **Figure 6.8**).
- ii) Responding dynamically to an application request about the service provisioning capability of the system according to its residual resources (as shown in **Figure 6.9**).
- iii) Handling simultaneous application requests and conflicts over these requests (as demonstrated in **Figure 6.10**).
- iv) Obtaining and displaying the status of multiple on-going application requests (as presented in **Figure 6.11**).

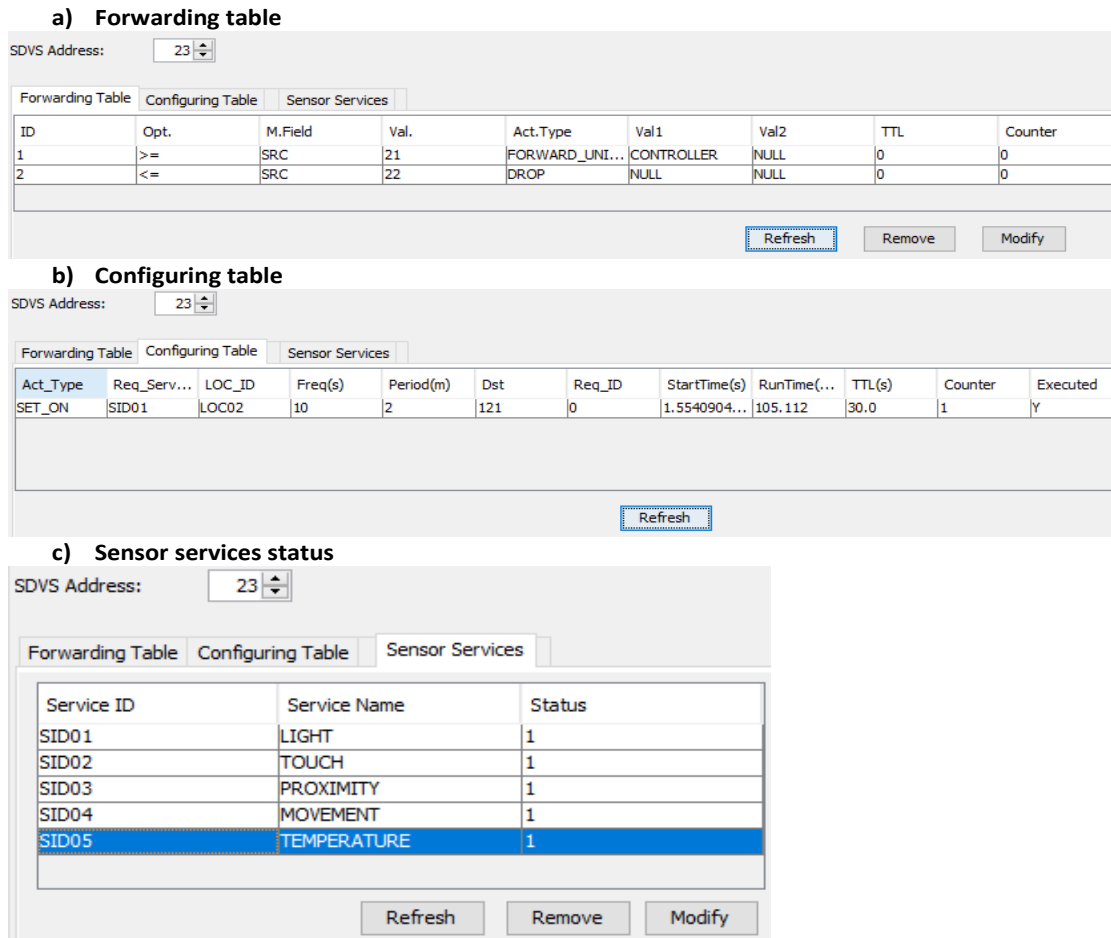


Figure 6.7 Status of the SDVS before its configuration

The programmable function of S-MANAGE is demonstrated in **Figure 6.7** and **Figure 6.8**. 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 6.7** and **Figure 6.8** 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. According to the forwarding instruction, the SDVS 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 6.7c**) to 0 (OFF) (as shown in **Figure 6.8c**).

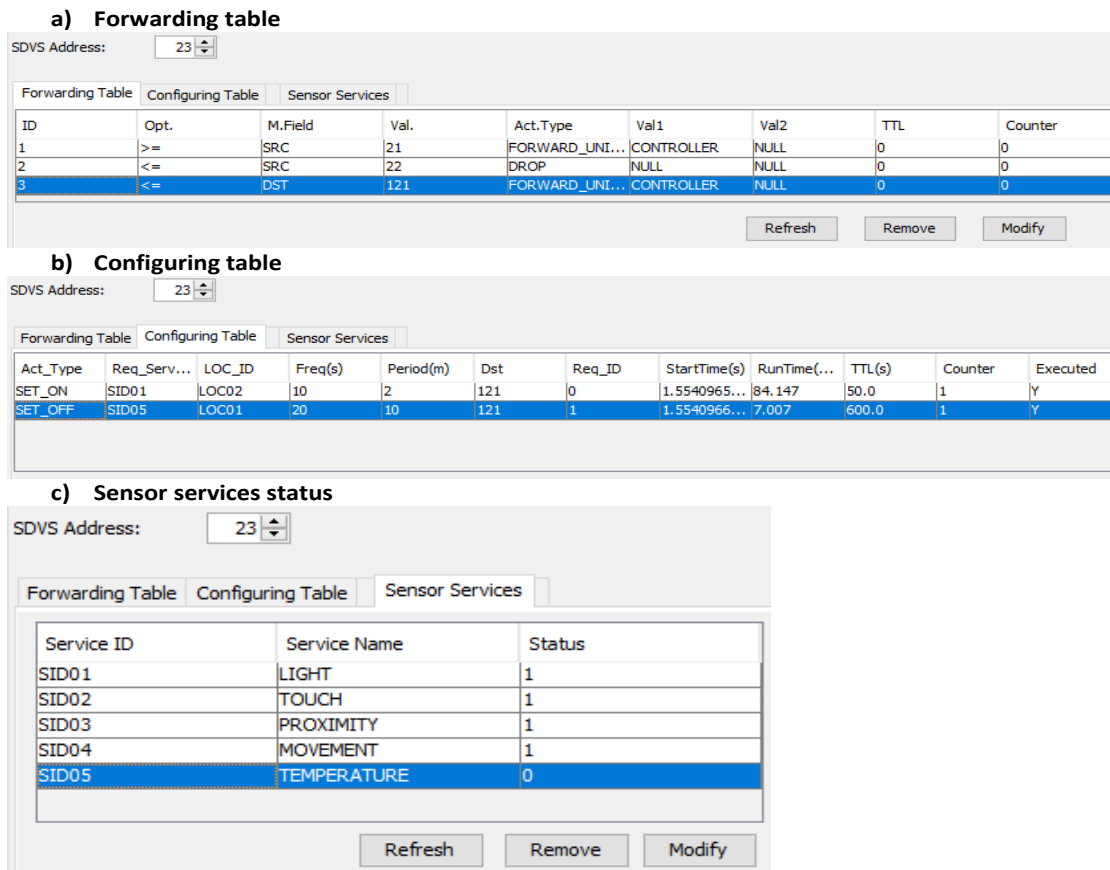


Figure 6.8 Status of the SDVS after its configuration

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 the 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 6.9**, the controller provides appropriate responses to the application request in the case i) it can fully achieve all the required services (see **Figure 6.9a**); ii) it partially achieves the required services and provides waiting time for obtaining the remaining required services (see **Figure 6.9b**), or is unable to provide the services because of insufficient resources (see **Figure 6.9c**).

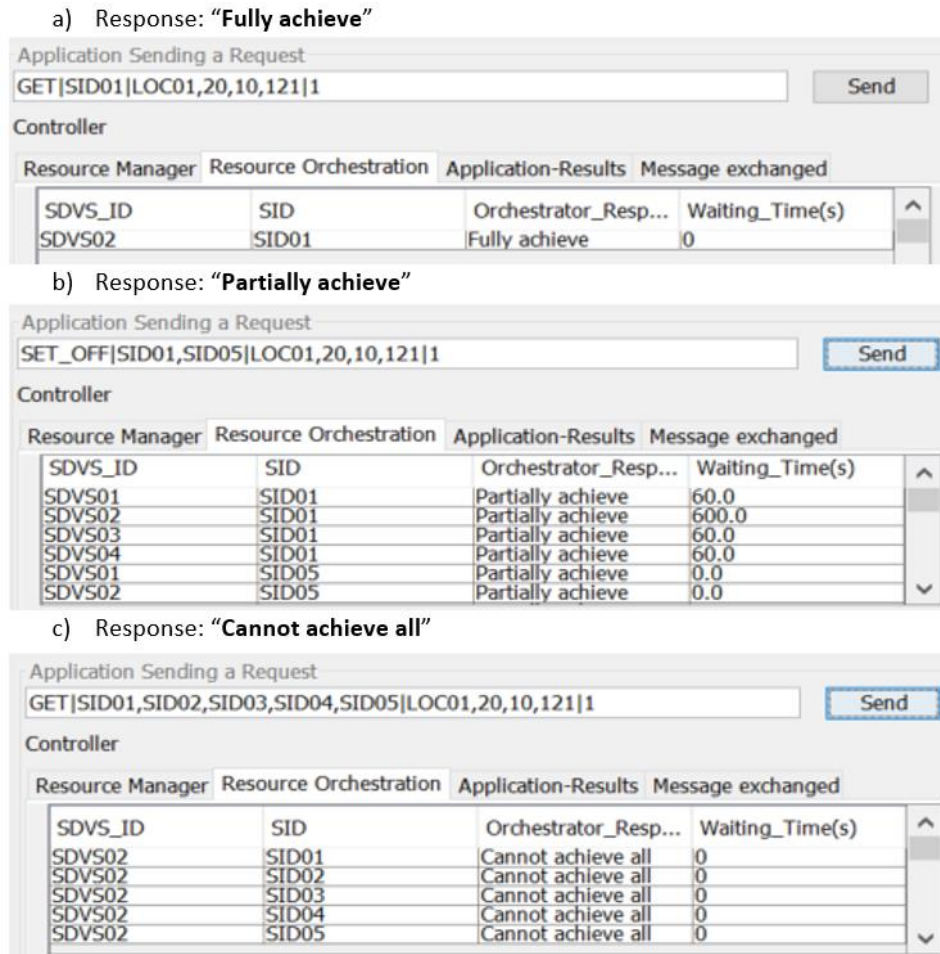


Figure 6.9 Dynamic response from the controller’s resource orchestrator to an IoT request

In addition, the system can handle multiple simultaneous application requests and resolve conflicts among these requests. As presented in **Figure 6.10**, 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 6.10** 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 two requests 2 and 3 for the same service. Request number 2 requires data from the sensor service, but request number 3 requires deactivating the sensor service. Therefore, SDVS02 delays request number 3 until it completes request number 1. In state 3, when request number 1 is done, SDVS02 achieves the required service for 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 6.10 Handling multiple application requests and solving conflicts among them

Figure 6.11 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 6.11 Status of ongoing application requests and corresponding results

The efficiency of the SD-IoT model is demonstrated by two performance metrics: the controller’s processing time, and the message overhead (as shown in **Figure 6.12**, **Figure 6.13**). The controller processing time represents the total time from when the controller receives an application request to when it sends out all configurations to required SDVSs. The average processing time of an application request depends on three parameters: i) the number of simultaneous application requests, ii) the similarities between demands from incoming requests and from previously processed requests, and iii) the availability of current SD-IoT resources. For example, the larger the number of simultaneous input requests, the higher the processing time is required for each request. Moreover, if the required resources are all available, the controller can process the requests immediately. Otherwise, the controller may ask the application if it is willing to wait until the required resources become available. This action also causes longer processing times. However, the processing time can be improved by reusing previously deployed configurations to satisfy new services.

Figure 6.12 presents the processing time of the model in response to one or multiple simultaneous application requests. The number of input requests is increased by 20 from 10 to 90. The requests are for one, two, three, four, or five services. The more services which are

required by a request or number of concurrent requests, the longer the processing time is needed. However, while the number of requests increases 9 times (10 to 90), the total processing increases about 3 times for all types of requests.

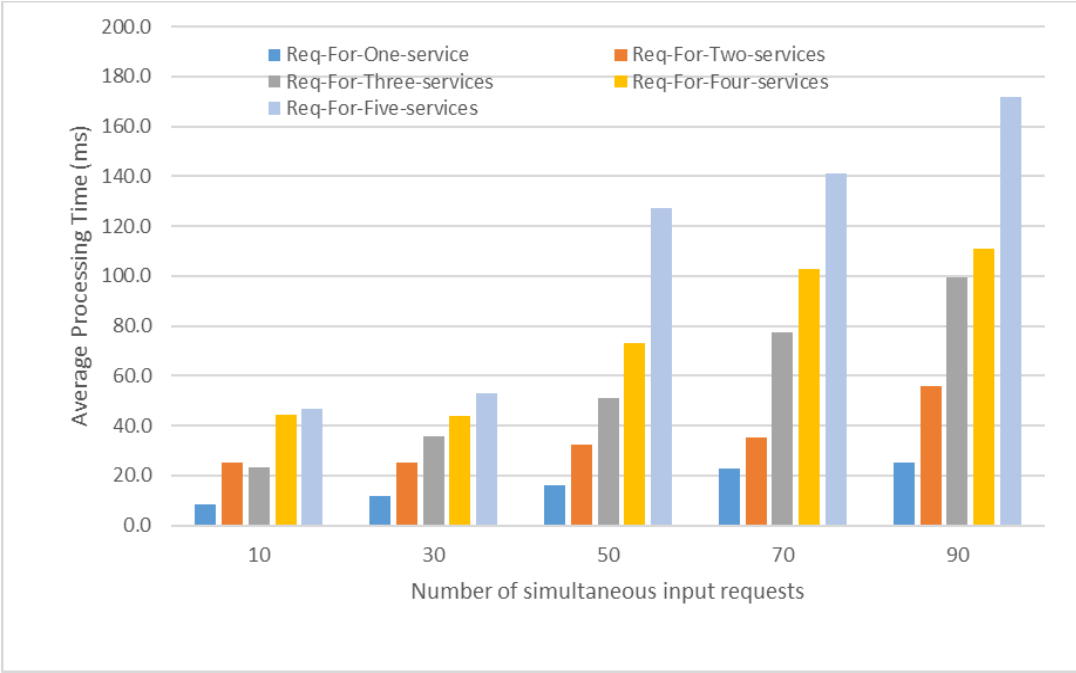


Figure 6.12 SD-IoTD Controller – Processing time for one per multiple simultaneous requests ranging between 10-90

We also examine messages overhead exchanged between the controller and its SDVSs. It is the total control and data messages needed for processing an application request. The number of control messages is reduced when the controller reuses the configuration of previous requests to provide services for a new request. **Figure 6.13** shows data and control messages for responding to requests. With a 5-times increase in the number of required services, the number of data messages increases about 35 times. Meanwhile, to achieve the results, the number of control messages increases about 2.5 times from two to five. This is because these requests are configured in the same manner. Moreover, the applications request for the same one, two, three, or four sensor services. A forwarding configuration can be applied for a flow of packets with similar

conditions. The number of data packets rises significantly since the achieved results must be sent in different periods to different destinations.

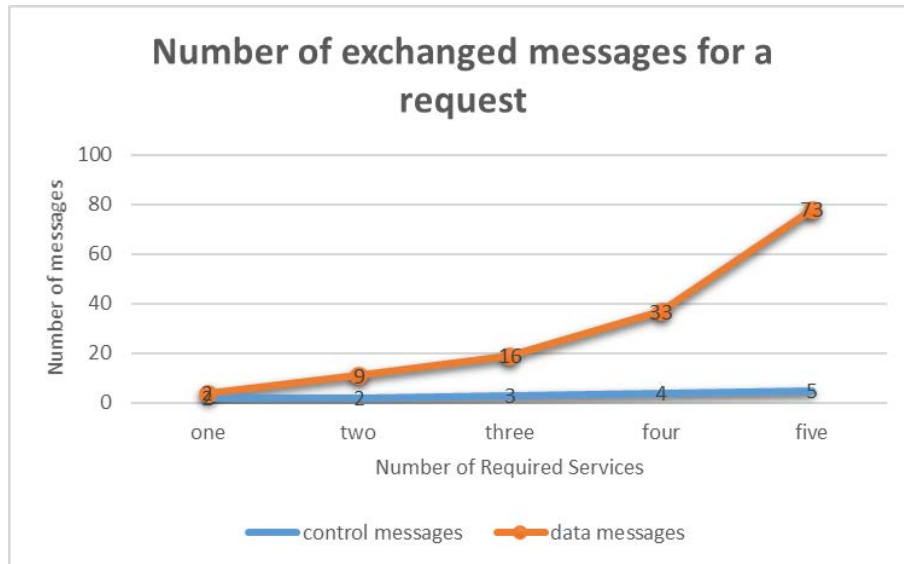


Figure 6.13 Number of exchanged control and data messages between SD-IoTD controller and SDVSs

6.7 Summary

In this chapter, we have introduced a software-defined IoT model with the proposed components, including SDVS (discussed in Chapter 4), S-MANAGE protocol (described in Chapter 5), and the SD-IoTD controller. We developed a new SD-IoTD controller that utilizes the S-MANAGE protocol to efficiently and flexibly orchestrate the SDVS resources in response to IoT demands. Detailed functional components, operational mechanisms and software design of the SD-IoTD controller have been provided. We presented the feasibility of the proposed model through the design and implementation of an SD-IoT platform prototype and evaluated the performance of the platform.

Chapter 7 Software-Defined Cluster Layer and LSSD-IoT Platform

7.1 Introduction

As discussed in Chapter 3, the proposed large-scale software-defined Internet of Things (LSSD-IoT) model provides a solution to the central control, management, and orchestration of geo-distributed IoT clusters. For the purpose of the control and management of such an LSSD-IoT model, we propose a software-defined (SD) cluster that handles the orchestration, coordination, and provision of IoT services. This chapter describes the proposed SD cluster and its operation in the LSSD-IoT platform (as depicted in **Figure 7.1**). In addition, this chapter demonstrates the implemented LSSD-IoT platform with all proposed components, including the SD-IoTC controller, SD-IoT model comprising SD-IoTD controller, S-MANAGE protocol, and SDVS. This work has been submitted to IEEE Transactions on Industrial Informatics journal and is under review.

The rest of this chapter is organized as follows. Section 7.2 presents the architecture and components of the SD cluster layer. Section 7.3 describes the operation of the LSSD-IoT platform in the provision of IoT services on demand. Section 7.4 discusses possible use cases of the LSSD-IoT platform. Section 7.5 presents the LSSD-IoT platform implementation. Section 7.6 demonstrates performance evaluation. Section 7.7 summarizes this chapter.

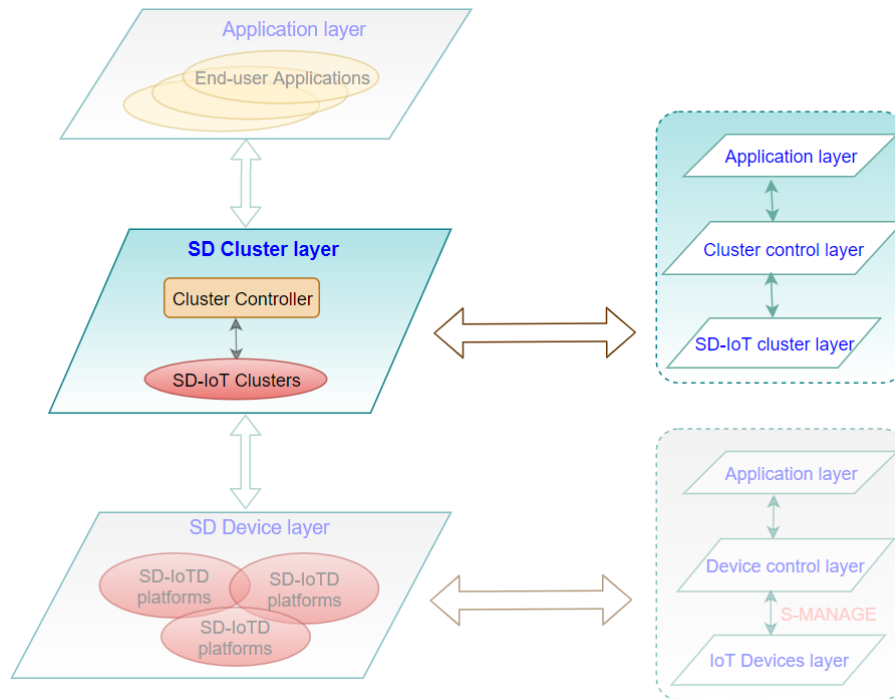


Figure 7.1 SD cluster layer in relation to LSSD-IoT model

7.2 SD Cluster Layer

This layer inherits much of the SDN architecture. It has 3 layers: the application layer, the Cluster control layer, and the SD-IoT cluster layer. The application layer is the same as the application layer of the overall LSSD-IoT, housing end-user applications. The Cluster control layer contains an SD-IoTC controller to perform both SDN functionality and IoT-specific service provisioning and coordinating functions. Instead of just SDN devices, they are replaced by SD-IoT clusters. Each cluster consists of an Open vSwitch and a host that represents an SD-IoT platform below. The host is termed as the SD-IoTC cluster. The OpenFlow and orchestration protocol are used for the communication between the SD-IoTC controller and SD-IoT clusters.

The architecture of this layer is highlighted within the LSSD-IoT model in **Figure 7.1**.

7.2.1 SD-IoTC Controller

This section describes the functional components of the SD-IoTC controller and provides mechanisms utilized by the SD-IoTC controller for the provision of IoT services on demand.

Architecture: In order to provision IoT services on demand, the SD-IoTC controller needs to not only understand IoT requests but also have a knowledge of all IoT devices under the control and management of the LSSD-IoT platform. To accomplish the functionality, the SD-IoTC controller is extended from a well-known Floodlight SDN controller. It communicates with its connected SD-IoT clusters. The SD-IoTC controller houses a set of components, as depicted in **Figure 7.2**. These functions allow the SD-IoTC controller to i) process IoT requests coming to the LSSD-IoT system, ii) control, manage, and orchestrate IoT clusters/devices, and iii) store temporary IoT services that can be shared between multiple IoT applications. Details of each component are as follows.

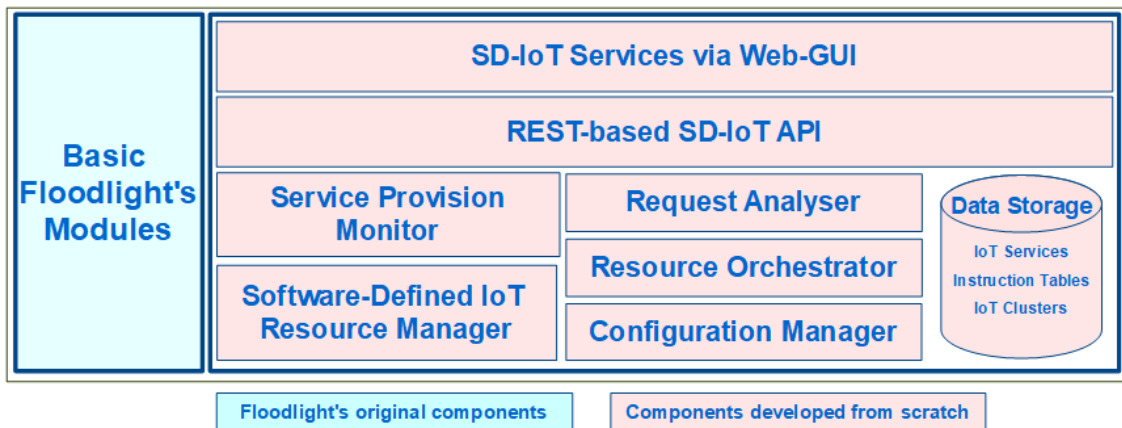


Figure 7.2 SD-IoTC controller architecture

SD-IoT Service via Web-GUI is an interface for users to specify IoT demands for the LSSD-IoT system. It also displays available IoT services and the status of IoT service provision.

REST-based SDIoT API is a Northbound Interface that provides abstractions of IoT resources to the application layer or users.

Request Analyser analyses input requests and provides specific requirements for the Resource Orchestrator.

Software-defined IoT Resource Manager is responsible for managing SD-IoT resources. It leverages SDN devices to update SD-IoT resources. The statistics are collected from other module applications (Basic Floodlight's Modules). It always updates connected SD-IoT clusters and their capabilities for service provision purposes.

Resource Orchestrator orchestrates SD-IoT clusters to provide required IoT services. In accordance with IoT requirements and available SD-IoT resources, it selects the most appropriate SD-IoT system to handle the IoT request. In addition, when an SD-IoT cluster cannot handle its assigned tasks, the Resource Orchestrator makes the best effort to re-schedule residual SD-IoT resources to satisfy the request.

Configuration Manager programs the core network and the SD-IoTD controller according to the instructions from the Resource Orchestrator. The Configuration Manager leverages the SDN resources to forward IoT requests to appropriate SD-IoTD systems. In accordance with the available SD-IoT clusters, it computes the flows for delivering IoT service requests as well as IoT service results to the intended destinations.

Data storage stores information concerning SD-IoT platforms connected to the LSSD-IoT system. The information includes locations of SD-IoT clusters, their capability such as provided services, and temporary data collected by the SD-IoT systems. The data is utilized by the Resource Orchestrator.

Service Provision Monitor watches the status of IoT service provision to release SDVSs involved in an IoT service provision and to update the status of SD-IoT resources. Moreover, it always checks the response from the SD-IoT cluster to see if they can handle IoT requests, and thus to announce the Resource Orchestrator to reschedule the task.

Orchestration mechanism: The SD-IoTC controller orchestrates services based on the availability of SD-IoT resources and a pool of already provisioned and available IoT services (**Algorithm 7.1**). This means that if an IoT request needs an IoT service that is currently

provisioned for another request, the controller reuses, if it can be accommodated, for the incoming request without further configuration on the underlying SD-IoT resources. In addition, if the SD-IoTC controller receives a response from an SD-IoTD controller that it cannot achieve the required services, the SD-IoTC controller re-orchestrates other SD-IoT clusters to handle the related requests. If several SD-IoT clusters are able to provide services to an IoT request, the controller allocates the task to the SD-IoT with the least number of tasks. The operation is illustrated via the pseudo-code below.

Algorithm 7.1: Resource Orchestration Mechanism

Input: a set of sdiot requests SRQ, incoming IoT request RQ, and updated SD-IoT resources RS

Output: a set of SD-IoT clusters (sdiot_name) and associated requests (sdiot_request)

Switch required_action (reqAct) in RQ **do**

case "GET":

for each required service (sid) in RQ **do**

 L1 = getSdiotReqWithGET(SRQ,sid,reqAct); //get a list of sdiot requests have the same requirements for sid and reqAct

if the size of L1 is 0**then**

 L2 = getListSdiotByAreaIdAndSid(list sdiot clusters, sid,areaId); //get a list of SD-IoT cluster can provide the sid in the required area AreaId

 PickBestSdiot (L2,sid)//get the SD-IoTD cluster with least task from L2

else if size of L1 is 1 **then**

 L3 = getSdiotWithGET(L1); //get the name of SD-IoT cluster in L1

else

 L4 = getListSdiotWithGET(L1)//get a list of SD-IoT cluster involved in the reqAct

 Best_Sdiot = PickBestSdiotWithLeastTask(L2); // pick the sdiot cluster with least

task from the database

for each sdiot_name in L4 **do**

if the sdiot_name is Best_Sdiot **do**

return (sdiot_name, sid)

end if

end for

end if

 get a list of (sdiot_name, sdiot requests)

end for

end if

case "SET_ON":

for each required service (sid) in RQ **do**

 L2 = getListSdiotByAreaIdAndSid(list sdiot clusters, sid,areaId); //get a list of SD-IoT

```

cluster can provide the sid in the required area
    Pick best sdiot from L2
    end for
get a list of (sdiot_name, sdiot requests)
case "SET_OFF":
    for each required service (sid) in RQ do
        L2 = getListSdiotByAreaIdAndSid(list sdiot clusters, sid,areaId);//get a list of SD-
IoT cluster can provide the sid in the required area
        Pick best sdiot from L2
    end for
get a list of (sdiot_name, sdiot requests)

```

7.2.2 SD-IoT Clusters and Communication with the SD-IoTC Controller

An SD-IoT cluster is composed of an SDN switch and a host representing an SD-IoT platform. SDN switches are networking devices that connect SD-IoT platforms to the LSSD-IoT system. They report on the connected IoT platforms. They allow the SD-IoTC controller to configure data flows between IoT clusters to deliver IoT requests to a proper SD-IoT platform or transmit returned results to data collection points. Hosts connected to SDN switches are representations of IoT clusters that are composed of required sensors/IoT devices. The hosts can be considered as SD-IoTC clusters that represent the underlying SD-IoT platform.

To orchestrate SD-IoT platforms, the SD-IoTC controller leverages both OpenFlow and an orchestration protocol. Via OpenFlow messages, the SD-IoTC controller updates network status and remotely configures forwarding functions of SDN switches for distributing IoT requests as well as forwarding results to the desired destinations. However, to update SD-IoT platforms connected to the LSSD-IoT system, the SD-IoTC controller leverages an orchestration protocol that is developed from a REST API. Via the protocol, the SD-IoTD controller of each SD-IoT platform provides its updated network statistics and available IoT services for the SD-IoTC controller.

7.3 LSSD-IoT Platform – Procedure of the Provision of IoT Services on Demand

The process of an SD-IoT service provision via the LSSD-IoT system includes four stages (as depicted in **Figure 7.3**). When receiving an IoT request, the SDN controller analyses the request and accordingly orchestrates needed IoT clusters to achieve the required services. The SD-IoTC controller also instructs the SD-IoTD controller on how to forward the results to IoT service collection points. In accordance with the configuration of the SD-IoTC controller, each engaged IoT cluster orchestrates its IoT devices to obtain the required services and then delivers achieved results to the expected destination.

- a) **Provision stage:** The SD-IoTC controller exposes its SD-IoT services via a REST-based northbound interface. It always listens to IoT interest and provides required SD-IoT services. The SD-IoTC controller exposes its SD-IoT services via the REST-based northbound API.
- b) **Orchestration stage:** The SD-IoTC controller analyzes the request and accordingly orchestrates available SD-IoT resources to achieve the required IoT services. It decomposes that request into one or multiple sub-requests that would be allocated to associated SD-IoT platforms.
- c) **Programmability stage:** SD-IoTD controller of each involved SD-IoTD system processes assigned tasks. It orchestrates its available resources to respond to the SD-IoTC controller if it can fully achieve the required services or not. If it can fully achieve the required services, it allocates the necessary tasks to SDVSs. Selected SDVSs program their represented underlying IoT devices to achieve the required services, then send results to the SD-IoTD controller. All achieved results are then sent to the desired data collection points. Otherwise, it responds to the SD-IoTC controller that it cannot fully achieve the required services. The SD-IoTC controller re-orchestrates residual SD-IoTD platforms to handle the request.

d) **Completion stage:** programmed IoT devices return results to the SD-LoTD controller, which then forwards them to required destinations. Whenever the SD-LoTC controller receives the announcement from applications that completely receive the required services, it relieves tasks of involved SD-LoT platforms.

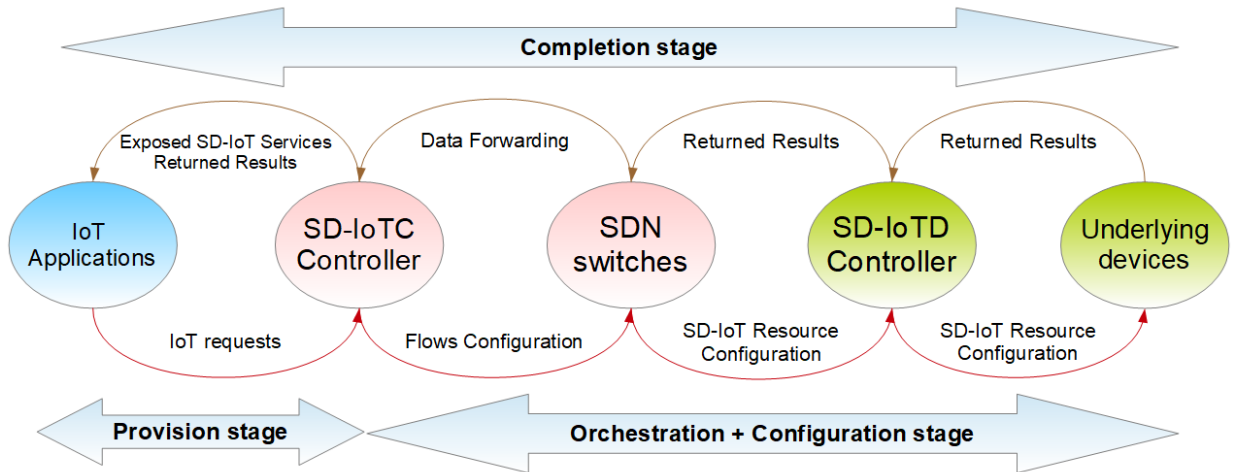


Figure 7.3 Overall procedure of provisioning IoT services on demand via LSSD-IoT platform

The procedure happens at the SD cluster and device layers. Details of the workflow at each layer are described below.

At SD cluster layer: The SD-LoTC controller analyses IoT demands and orchestrates the requests in accordance with the capability of the LSSD-IoT system. The operation of the SD-LoTC controller is illustrated in **Figure 7.4**.

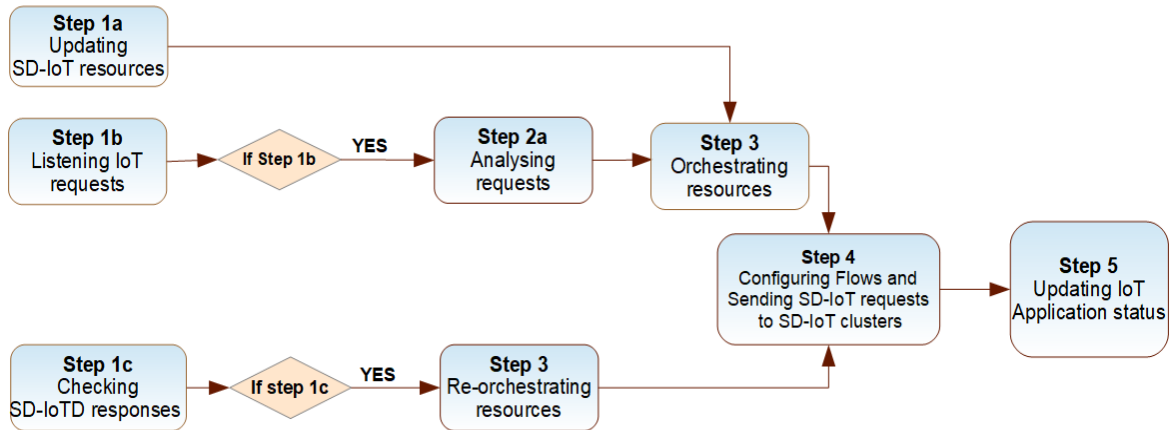


Figure 7.4 Workflow of the SD-LoTC Controller

At the SD device layer: Upon the requests from the SD-LoTC controller, the SD-LoTD orchestrates its available resources to achieve the required services. The operation of the SD-LoTD controller is depicted as in **Figure 7.5**.

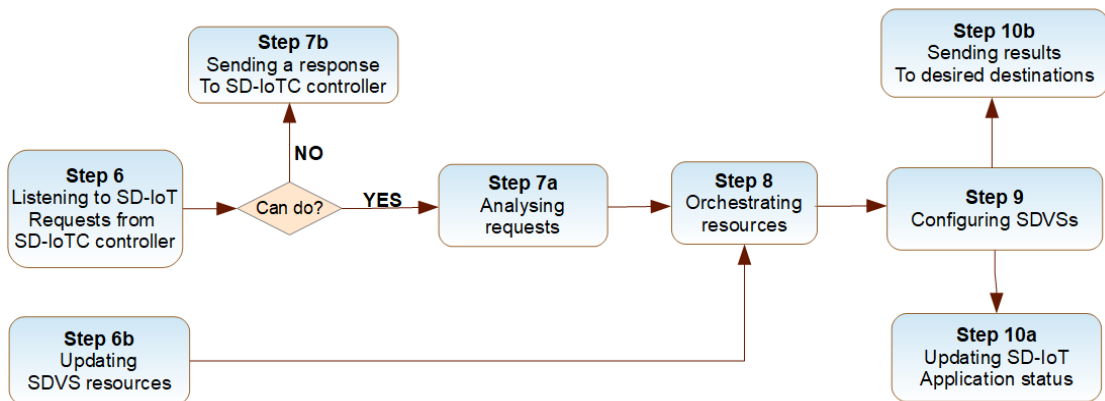


Figure 7.5 Workflow of the SD-LoTD Controller

7.4 LSSD-IoT Platform – Use cases

According to a survey in 2019 [121], IoT applications are classified into six categories, including smart city, healthcare, commercial, environmental, general aspects, and industrial. Among the groups, smart city accounts for the highest portion with 29%, followed by healthcare

and commercial, with 20% and 14%, respectively. Environmental applications have 12%. Among these applications, three groups of IoT applications that highly demand large-scale IoT infrastructure are smart logistics/cities, healthcare, and environmental monitoring [6]. We take some real IoT use cases as examples to illustrate the operation of the proposed LSSD-IoT model to provision of IoT services on demand; from that, we develop an implementation scenario.

❖ **Use case number 1: Monitoring Air Pollution**

Air pollution has become one of the controversial concerns around the world. Therefore, there is a demand for estimating air pollution levels over a wide area where people experience diseases related to the polluted environment in a city A. To achieve the estimated values, there need to be measurements associated with Carbon dioxide (CO₂) which is emitted from electricity generation, factories, and vehicles; Carbon Monoxide (CO) that is emitted from vehicle exhausts and is formed when carbon fuels are not burned completely; and Nitrogen dioxide (NO₂) that is emitted from motor vehicle exhausts [122]. To collect these values, we need the involvement of sensing systems from i) factories or big building that provide measurements about CO₂, street monitors that provide values about CO, and NO₂ (**Figure 7.6**). However, the sensing systems are distributed over a large scale, so the collected values are transferred to a central point in the cloud.

Requirements from the use case number 1: IoT services, for example; CO₂, NO₂, and CO value from different IoT systems within the city A are collected within a required period. The ultimate goal of the IoT application is to get the average value of the three readings. Therefore, these required services are collected and aggregated at the cluster head of each IoT cluster, which sends computed values to the SDN controller for further processing. The SDN controller orchestrates distributed IoT systems to deliver their services to the required destination. To achieve it, the SDN controller configures not only SDN switches that connected to IoT systems but also IoT devices within these IoT systems.

IoT service provision scenario number 1 (Figure 7.7): measurements about CO₂, NO₂, and CO are represented by services s₁, s₂, and s₃, respectively. The service s₁ is provided by the IoT cluster 1 and 2. The services s₂ and s₃ are provided by the IoT cluster 3, 4, and 5. The cluster

head of each cluster collects the required services and sends them to desired data collection points.

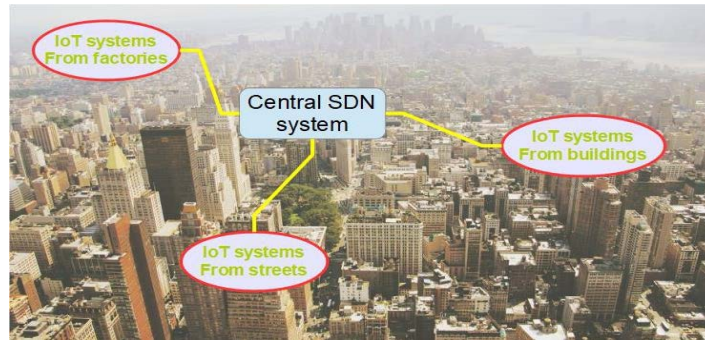


Figure 7.6 Monitoring air pollution use case

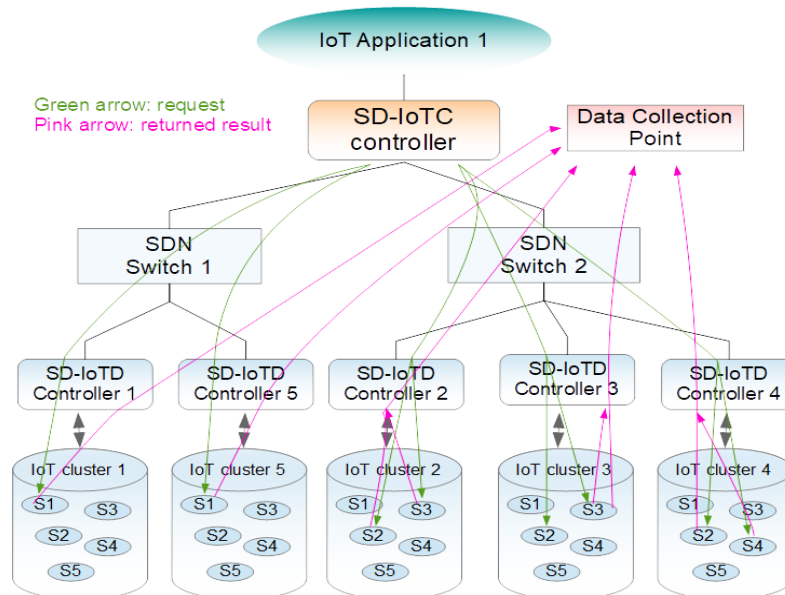


Figure 7.7 IoT service provision scenario number 1

❖ **Use case number 2: Smart Traffic Control**

Take a smart control of traffic flow in Los Angeles, for example [123], it demands real-time readings as well as actuating from sensors of the traffic control platform. Real-data from road-surface sensors and closed-circuit television cameras are collected and sent to a central traffic

management platform in the cloud to update the status of traffic flow. By analyzing the data, the platform notifies users about traffic congestion and signal malfunctions. In addition, a network of smart controllers is developed to make instant adjustment of traffic light conditions second-by-second.

Requirements from the use case number 2: the application needs not only real-time data from IoT systems but also instant actions on engaged IoT systems. Therefore, to provide IoT services for the use case number 2, three SD-IoT clusters are needed for collecting data from the traffic systems, and one system for taking actions based on collecting data (**Figure 7.8**). All sensing data are aggregated at their SD-IoTD controller. The SD-IoTD controller configures SDVSs to collect road-surface and closed-circuit television cameras readings. The collected data are sent to the SD-IoTD controller of each cluster, then the SD-IoTD controller aggregates the data and sends the average values to the SDN controller according to routing paths specified by the SDN controller. The SDN controller analyses the achieved results in order to make instant adjustments on engaged traffic lights. The required configuration is sent to the required SD-IoTD controller that accordingly configures their underlying IoT devices to control the traffic light.

IoT service provision scenario number 2 (Figure 7.9): the road-surface and closed-circuit television are represented by the service s4 and s5, while the traffic light is presented by the service s6. These services are collected from IoT clusters 3, 4, and 5. The service s4 and s5 are aggregated at the cluster head and then sent to the central controller, while the service s6 are configured by the cluster head under the control of the central controller.

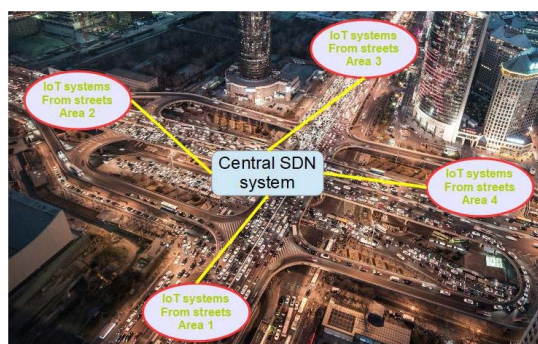


Figure 7.8 Smart traffic control use case

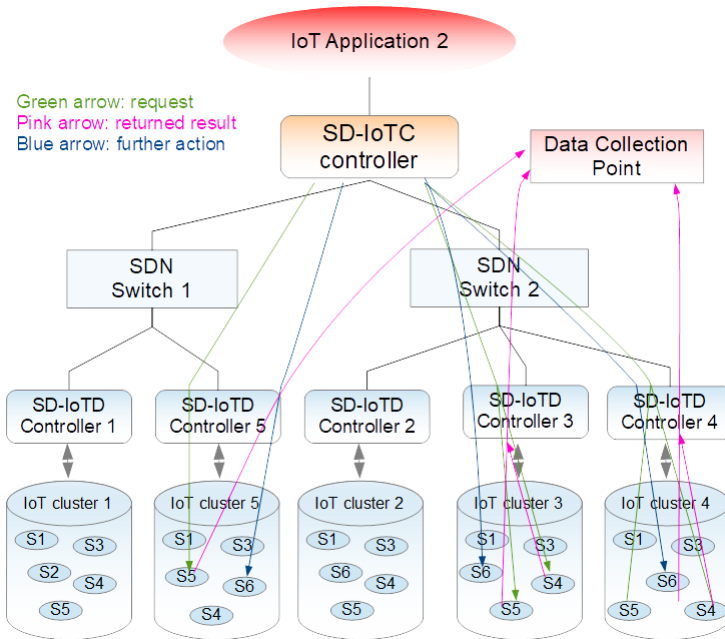


Figure 7.9 IoT service provision scenario number 2

❖ **Use case number 3: Train Load Management**

Another real use case is about the combination of IoT data from several IoT systems to build a smart train system in London [123]. The development aims to project the load of train passenger cars going out of and in to the city, some train operators have aggregated data from CCTV cameras, movement sensors, and ticket sales deployed along with the platform. According to analyzed data, they can estimate how much load for each car, then advise the passengers to spread along the train for balancing the load. Thanks for that, train delays can be prevented.

Requirement of the use case number 3: the train management system at each station needs information about a load of the incoming trains so it can advise passengers to spread out along with the platform. Thus, the central SDN controller of the whole train systems needs to collect the load of all involved trains coming to the station. Then, it sends the collected values to the management system at the station that make associated announcements to passengers there to do the right actions.

Provision scenario number 3 (Figure 7.10): the television camera, ticket sale, and movement are represented by services s5, s7, and s8, respectively. S5 is collected from the IoT cluster 1 to 5 and sent to the cluster head. S7 is gathered by the IoT cluster 5 and sent to the central SDN controller which forwards it to the cluster heads of IoT cluster 1 to 4. S8 is aggregated by the IoT cluster 1 to 4. The cluster heads of the IoT cluster 1 to 4 make appropriate announcements according to aggregated information.

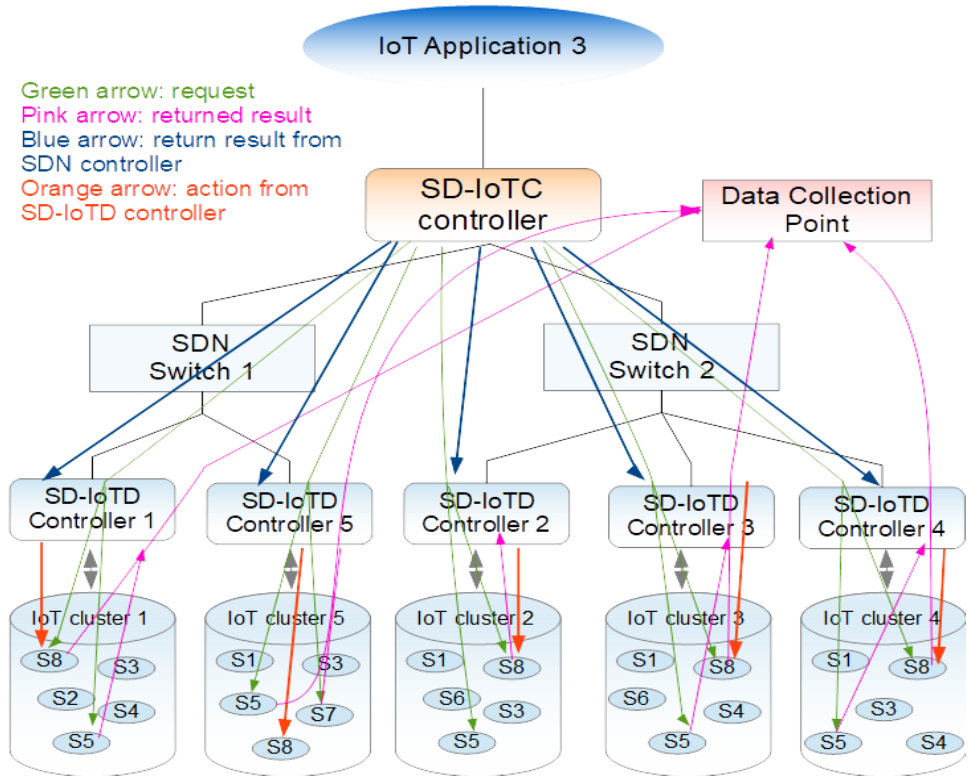


Figure 7.10 IoT service provision scenario number 3

Details to IoT services needed by each use case are presented in **Table 7.1**.

Table 7.1 Summary of IoT application requests and corresponding IoT services provided by IoT clusters (R_i represents a request from an IoT use case)

IoT service	Building 1 (factory/train station)	Building 2 (train station)	Street monitoring systems (or train system)		
IoT services	IoT cluster 1	IoT cluster 2	IoT cluster 3	IoT cluster 4	IoT cluster 5
S1 (CO ₂)	R1	R1			
S2 (NO ₂)			R1	R1	R1
S3 (CO)			R1	R1	R1
S4 (road-surface)			R2	R2	
S5 (television camera)	R3	R3	R2, R3	R2, R3	R2, R3
S6 (traffic light actuator)			R2	R2	R2
S7 (movement)					R3
S8 (ticket sale-central management)	R3	R3	R3	R3	

7.5 LSSD-IoT Platform Implementation

7.5.1 Implemented Platform

In previous sections, we present the proposed LSSD-IoT model, describe the architecture, and the design of components and their specific functionalities. In this section, we present the implemented platform that integrates both cluster and device layers to form the large-scale IoT on-demand service platform.

SD-IoTC Controller: We have designed and implemented SD-IoTC and its components to interpret application requests, orchestrate, provision, coordinate services over clusters of IoTs under its control.

An application interface: The interface enables communication between users/application developers and the proposed LSSD-IoT platform. It presents available IoT services to users as well as allows users to specify their requests for the IoT services.

SD-IoTC clusters: Entities that represent individual IoT clusters. This includes an information base that holds the knowledge of its local IoT environment, resources, usage and communication protocol for communication with SD-IoTC controllers. It is also part of the SD-IoTD controller to understand the specific sub-services required by its local SD-IoT platform.

SD-IoTD Controller: In Chapter 6, we have designed and implemented SD-IoTD and its components to interpret application requests, orchestrate, provision, coordinate services over devices in an IoT cluster. An SD-IoTD system represents one IoT cluster that comprises a number of sensors/IoT devices within a small area, e.g., a street area, a building, or a campus.

Coordination protocol for orchestrating sub-requests between the SD-IoTC controller and an SD-IoTC cluster.

At the cluster level, there is a user interface for users entering requests and four tables providing information regarding i) available SD-IoT resources, ii) requirements of IoT requests, iii) acknowledgment from IoT applications, and iv) results of resource orchestration and execution status.

At the device level, there is also a user interface for receiving users demands or requests from the SD-IoTC controller; and four tables presenting i) available SDVSs, ii) results of resource orchestration, iii) the number of messages exchanged between each SDVS and the SD-IoTD controller, and iv) SDVS's forwarding table, configuring table and sensor services.

S-MANAGE protocol for the communication and management between the SD-IoTD controller and its underlying sensors. This work has been presented in Chapter 5.

SDVS (Software-defined virtual sensor): This component has been designed to represent the underlying regional sensors and enrich their capabilities for programming, configuring, and provisioning services on demand within the local cluster. Via SDVSs, each SD-IoTD system musters the capability of its represented IoT cluster. The SD-IoTD has knowledge of all IoT

connected devices' attributes in terms of available IoT services, networks, and limitations. The SDVS possesses software drivers and plug-in interfaces for various types of underlying physical sensors/IoT devices. This has been presented in Chapter 4.

Floodlight SDN controller (open source): Floodlight is a Java-based controller that allows developers to easily develop applications or integrate new functional components from/to the fundamental architecture. We have implemented the SD-IoTC controller as a new application module within the Floodlight.

OpenFlow protocol (open source): This is a well-known open communication protocol that allows the SDN controller to program SDN switches as well as change the network configuration. Taking advantage of that, the SD-IoTC controller can configure data flows to distribute IoT requests to appropriate SD-IoTC clusters and to forward results to desired data collection points.

OpenFlow switch (open source): This is an Open vSwitch (OVS) that implements OpenFlow protocol as a southbound interface. It contains a switch's basic information such as IP address, port names, and port numbers. Moreover, it has a data plane element that performs packets forwarding. Through the switches, the SD-IoTC controller can update its network of connected SD-IoTC platforms through the switches and can compute routing paths based on the knowledge of the underlying platforms.

The structure of all the components is shown in **Figure 7.11**.

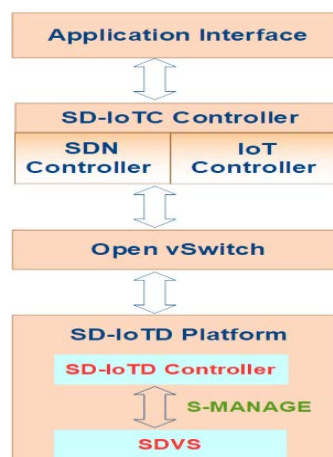


Figure 7.11 Detailed implementation of LSSD-IoT architecture

7.5.3 Implementation Scenario

To demonstrate the application of the proposed LSSD-IoT model, we develop a use case scenario that requires IoT services on demand (as shown in **Figure 7.12**). In the scenario, there are five local locations distributed geographically over a wide area, A. An application needs to be developed to provide the weather condition of these locations, which are located in three areas A1, A2, A3. Ideally, each location should be served by a local IoT system which is more responsive to deal with specific local issues concerning resources, response time, and mobility. LSSD-IoT system is designed to address such an on-demand service by orchestrating and distributing resources and sub-services appropriately to locations as needed. The LSSD-IoT system, in this case, may be divided into 5 local IoT subsystems as follows: IoT systems 1 and 2 are in A1. IoT systems 2 and 4 are in A2. IoT system 5 is in A3. The LSSD-IoT system orchestrates these IoT subsystems to satisfy the demands from the current application, and if new applications are to be provisioned (which may be other services other than weather condition, for example, pollution condition), the LSSD-IoT will schedule available shared resources from all IoT subsystems to satisfy the new applications demand according to the location and capability of the subsystems.

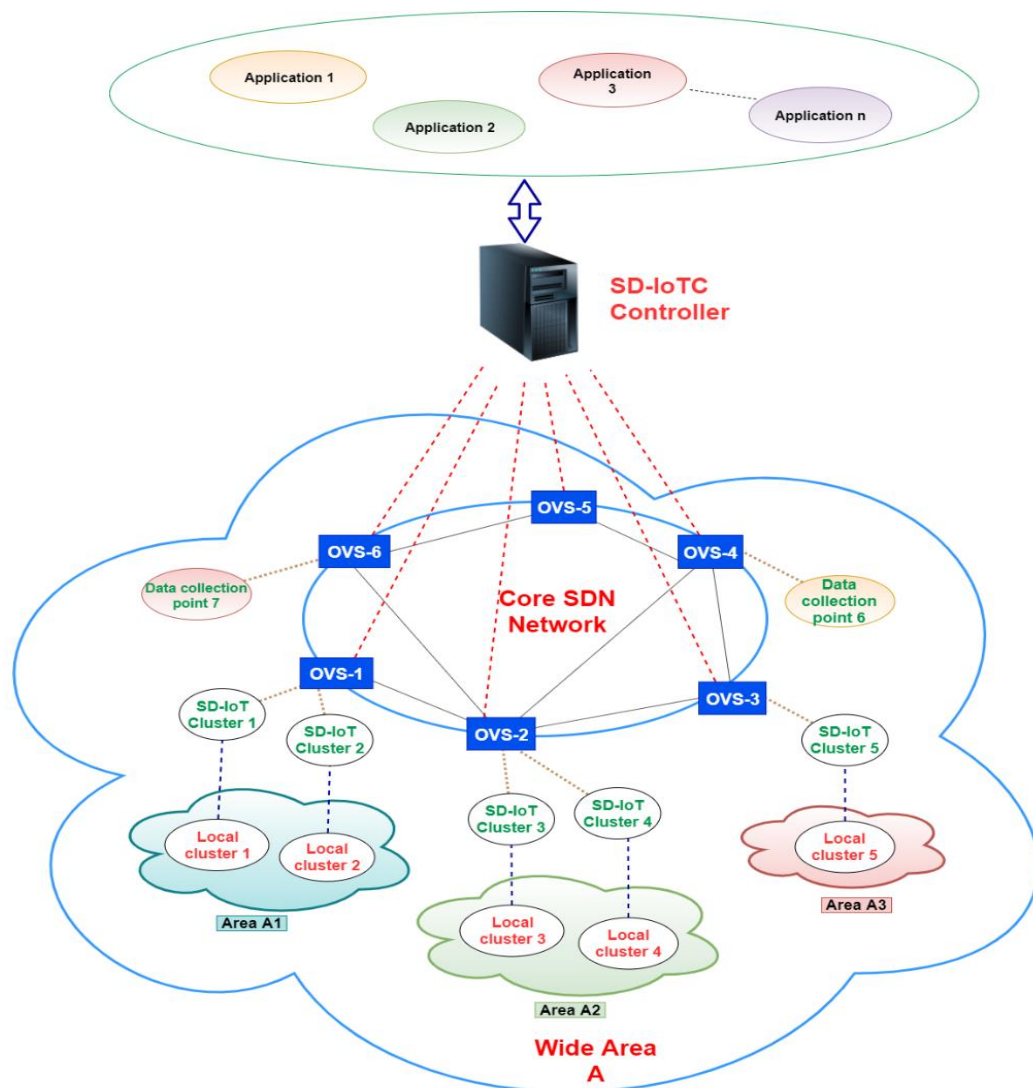


Figure 7.12 LSSD-IoT Model - Implementation scenario

7.5.3 Implementation Set up

For demonstration, we implement a prototype, as presented in **Figure 7.13**. All components of the proposed architecture are built in software. In order to establish the LSSD-IoT prototype, we implement i) an SD-LoTC controller, which is extended from the Floodlight SDN controller

to handle the SDN network of IoT clusters; ii) a network of OpenFlow switches where IoT clusters are connected, and iii) seven hosts that house local IoT clusters or data collection storage. Five SD-LoTD systems run on five hosts, and two IoT storage applications run on the remaining hosts. Each SD-LoTD platform contains a network of five SDVs, each of which represents five sensor types.

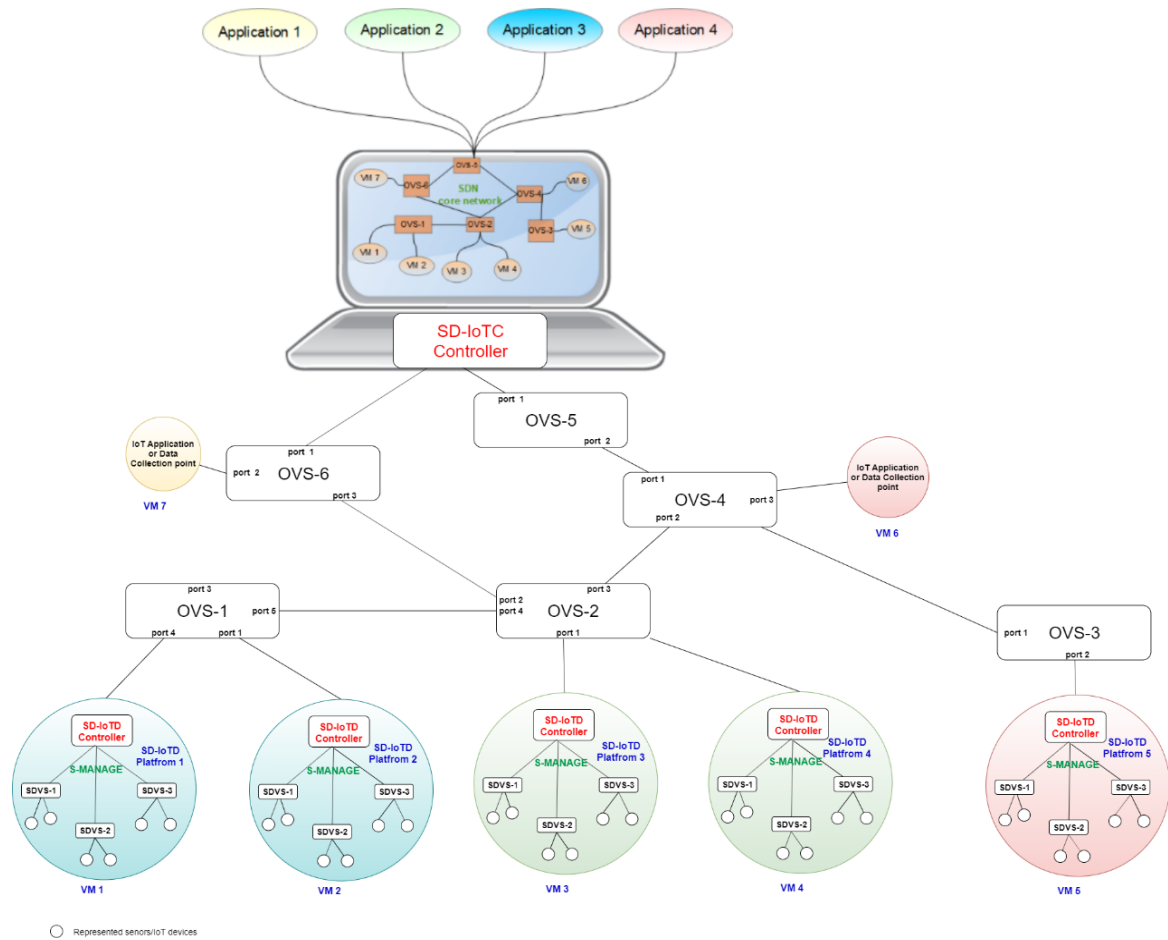


Figure 7.13 Detailed implementation of the LSSD-IoT platform

Details of implementation set up are as follows:

- + All the main components are built on one PC running Ubuntu 16.04 LTS with detailed configuration: Memory:16GB; Processor: IntelR CoreTM i7-7600U CPU @ 2.8GHz x 4; OS type:64-bit; Disk:235GB.

- ✚ A Web-GUI is developed by leveraging the bootstrap container based on the Spring framework *(<http://projects.spring.io/spring-framework/>). Users can request for IoT services by accessing the GUI via a link < controller IP address>:8080/ui/pages/sdiot.html.
- ✚ The SD-IoTC controller is extended from the Floodlight SDN controller, a well-known open platform for controlling and managing SDN devices. The SD-IoTC controller is developed and run in Eclipse Java EE IDE, version: Photon Release (4.8.0).
- ✚ A network of SDN devices is built in the Mininet simulator. In our implementation, we use Open vSwitch 2.5.5 and OpenFlow 1.3 (0x04).
- ✚ SD-IoTD systems are a Java-based platform developed in NetBeans 8.2. These systems are deployed in Mininet hosts connected to SDN devices. It is connected to its own database built by MySQL. We have implemented three main software components of the proposed SD-IoTD model: the SD-IoTD controller, the S-MANAGE protocol, and the SDVS.
 - The control module includes classes responsible for the controller’s functionality.
 - The southbound interface module is composed of classes for the construction of S-MANAGE messages, forwarding tables, and configuring tables of SDVSs.
 - The virtual representation module contains classes for initiating instances of an SDVS and its software-defined functions.
- ✚ The databases for SD-IoTC controller and SD-IoTD controllers are built by using MySQL, version 5.7.27-0ubuntu0-16.04.1.

7.5.4 SD-IoTC Controller – Software Implementation

This section describes the software component of the SD-IoTC controller.

❖ Software Implementation

The SD-IoTC controller is extended from the Floodlight controller. This section presents software elements that perform the above-discussed features of the SD-IoTC controller.

The SD-IoTC controller is implemented as an application module in the Floodlight platform. The main classes of the SD-IoTC controller are shown in **Figure 7.14**. The SdiotModule class contains main classes that perform the function of the SD-IoTC controller. Details of the

RequestAnalyser and SdiotModule classes are illustrated in **Figure 7.15** and **Figure 7.16**, respectively.

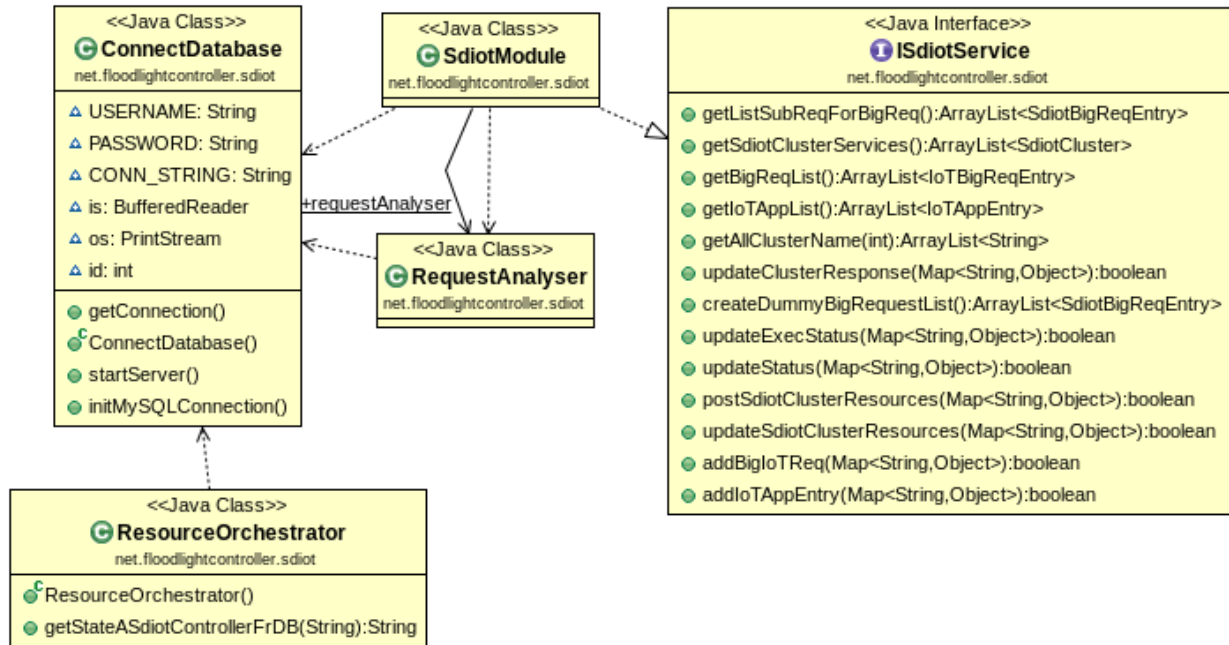


Figure 7.14 Class diagram of the SD-IO TC controller

```

<<Java Class>>
RequestAnalyser
net.floodlightcontroller.sdiot


theBestSdiotForEachSID: String

RequestAnalyser()
composeSubReqToSdiotController(String):String
decomposeBigRequestIntoSubReqList(String):ArrayList<String>
decomposeBigReqIntoSubReqListWithouSmart(String,int):ArrayList<String>
decomposeBigReqIntoSubReqList(String,int):ArrayList<String>
decomposeBigRequestWithBigReqIDIntoSubReqListBySIDUsingDatabase(String,int):ArrayList<String>
decomposeBigReqIntoSubReqListUsingDatabaseForCaseGET(ArrayList<SdiotBigReqEntry>,String,int):ArrayList<String>
decomposeBigReqIntoSubReqListUsingDatabaseForCaseSET(String,int):ArrayList<String>
decomposeBigRequestWithBigReqIDIntoSubReqListBySIDUsingAvailableSdiotResources(ArrayList<SdiotCluster>,String,int):ArrayList<String>
decomposeBigReqIntoSubReqListBySIDUseAvailSdiotResources(ArrayList<SdiotCluster>,String,int):ArrayList<String>
decomposeBigReqIntoSubReqListBySIDUseAvailSdiotResourcesForSmartOrch(ArrayList<SdiotCluster>,ArrayList<SdiotBigReqEntry>,String,int):ArrayList<String>
decomposeBigReqIntoSubReqListBySIDUsingAvailableSdiotResources(ArrayList<SdiotCluster>,ArrayList<SdiotBigReqEntry>,String,int):ArrayList<String>
getSdiotReqWithGET(ArrayList<SdiotBigReqEntry>,String,String):ArrayList<SdiotBigReqEntry>
pickBestSdiotClusterNameWithLeastState(ArrayList<String>):String
getListSdiotNameWithGET(ArrayList<SdiotBigReqEntry>):ArrayList<String>
getSidListByClusterNameFrDatabase(String):ArrayList<String>
getToTClusterListBySID(String):ArrayList<String>
getToTClusterListByLocIdFrDatabase(String):ArrayList<String>
getToTClusterIpAddrListByLocIdFrDatabase(String):ArrayList<String>
getIPAddrByClusterNameFrDatabase(String):String
getPairsOfSdiotIpAddrAndSubRequest(ArrayList<String>):ArrayList<String>
getListSdiotNameByArealdAndSidFrDatabase(String,String):ArrayList<String>
getListSdiotNameByArealdAndSidFrSdiotClusterResource(ArrayList<SdiotCluster>,String,String):ArrayList<String>
pickBestSdiotFromListSdiotSID(String,String,ArrayList<String>):String
produceBigRequestFromSubRequest(String):String
rescheduleABigRequestUsingDatabase(String,String):ArrayList<String>
rescheduleABigRequestUsingAvailableSdiotResource(ArrayList<SdiotCluster>,String,String):ArrayList<String>
removeSdiotNameFromPotentialSdiotList(ArrayList<String>,String):ArrayList<String>
removeClusterWithResponseCANNOTFrBigReqList(ArrayList<SdiotBigReqEntry>,String,String):ArrayList<SdiotBigReqEntry>
addNewSubReqToBigReqList(ArrayList<SdiotBigReqEntry>,String,ArrayList<String>):ArrayList<SdiotBigReqEntry>
getToTClusterListByLocIdFrSdiotClusterResource(ArrayList<SdiotCluster>,String):ArrayList<String>
getSdiotClusterAndSidWithGET(ArrayList<SdiotBigReqEntry>):ArrayList<String>
sendResultToMininetHost(String,String,String,int):void
getIpAddrDstHost(String):String











































```

Figure 7.15 RequestAnalyser class

<<Java Class>>

 **SdiotModule**

net.floodlightcontroller.sdiot

-  SdiotModule()
-  getName():String
-  isCallbackOrderingPrereq(OFType,String):boolean
-  isCallbackOrderingPostreq(OFType,String):boolean
-  getModuleServices():Collection<Class<? extends IFloodlightService>>
-  getServiceImpls():Map<Class<? extends IFloodlightService>,IFloodlightService>
-  getModuleDependencies():Collection<Class<? extends IFloodlightService>>
-  init(FloodlightModuleContext):void
-  receive(IOFSwitch,OFMessage,FloodlightContext):Command
-  startUp(FloodlightModuleContext):void
-  getRequestInByteArray():byte[]
-  getRequestInString():String
-  getInputForTcpServerSocket():String
-  sendPacketOutMessage(byte[],IOFSwitch):void
-  sendSubRequestToSdiotUsingPktOutInUdp(byte[],IOFSwitch,String,String):void
-  sendSubRequestToSdiotUsingPktOutInTcp(byte[],IOFSwitch,String,String):void
-  startListeningToOneBigRequest():void
-  startListeningToMultiBigReqWitBigReqID():void
-  startListeningToMultiBigReqWitBigReqIDPUTTOAPI():void
-  exeloTBigRequestFrGUI(String):void
-  exeloTBigRequestFrGUIWithSmartOrch(String):void
-  exeloTBigRequestFrGUIVersion1(String):void
-  exeloTBigRequestFrGUIWithoutSmartOrch(String):void
-  exeloTBigRequestFrGUIConcerningCurrIoTReq(String):void
-  startCheckingAndReschedulingSubReqFromSdiotUsingDatabase():void
-  startCheckingAndReschedulingSubReqFromSdiotUsingAvailableSdiot():void
-  buildASDIoTReqEntry(String,String):SdiotBigReqEntry
-  buildASDIoTReqEntryForSmartOrch(String,String):SdiotBigReqEntry
-  startListeningToSocketForSendingPktOutMessage():void
-  startTCPSocketServer():void
-  sendPostRequestForSwitchInfo(String,String):JSONArray
-  sendPostRequestForDeviceInfo(String,String):JSONArray
-  getSwitchesFromFloodlight(String):JSONArray
-  getDevicesFromFloodlight(String):JSONArray
-  getSwitchHostList(String,String):ArrayList<String>
-  sendSubRequestToSdiotUsingUDPSocket(String,String,int):void
-  sendSubRequestToSdiotUsingTCPSocket(String):void
-  getListSubReqForBigReq():ArrayList<SdiotBigReqEntry>
-  updateClusterResponse(Map<String,Object>):boolean
-  updateExecStatus(Map<String,Object>):boolean
-  postSdiotClusterResources(Map<String,Object>):boolean
-  updateSdiotClusterResources(Map<String,Object>):boolean
- getAllClusterName(int):ArrayList<String>

```

● setListBigReq(ArrayList<SdiotBigReqEntry>):void
● createDummyBigRequestList():ArrayList<SdiotBigReqEntry>
● updateBigRequestEntry(Map<String,Object>):boolean
● findIndexByID(int):int
● findIndexByIDAndClusterName(int,String,String):int
● checkClusterResponse(ArrayList<SdiotBigReqEntry>):ArrayList<String>
● getSdiotClusterServices():ArrayList<SdiotCluster>
● getBigReqList():ArrayList<IoTBigReqEntry>
● addBigIoTReq(Map<String,Object>):boolean
● addIoTAppEntry(Map<String,Object>):boolean
● getIoTAppList():ArrayList<IoTAppEntry>
● updateClusterStateToDatabaseUsingInfoFrAvailableResources(String):void
● updateClusterStateToDatabaseUsingInfoFrMysql(String):void
● updateToMysql(int,String):void
● updateStatus(Map<String,Object>):boolean
● checkCompletionStatusOfSdiotReq():void
● checkExecutionStatusOfSdiotReq():void
● countNumOfSdiotReqPerBigReq(String,ArrayList<SdiotBigReqEntry>):int
● countNumOfSdiotReqsCompletelyDone(String,ArrayList<SdiotBigReqEntry>):int
● countNumOfSdiotReqForAnIoTReqsExecuted(String,ArrayList<SdiotBigReqEntry>):int

```

Figure 7.16 SdiotModule class

To expose the module to other modules inside the Floodlight controller, we implement the REST-API interface, as shown in **Figure 7.17**.

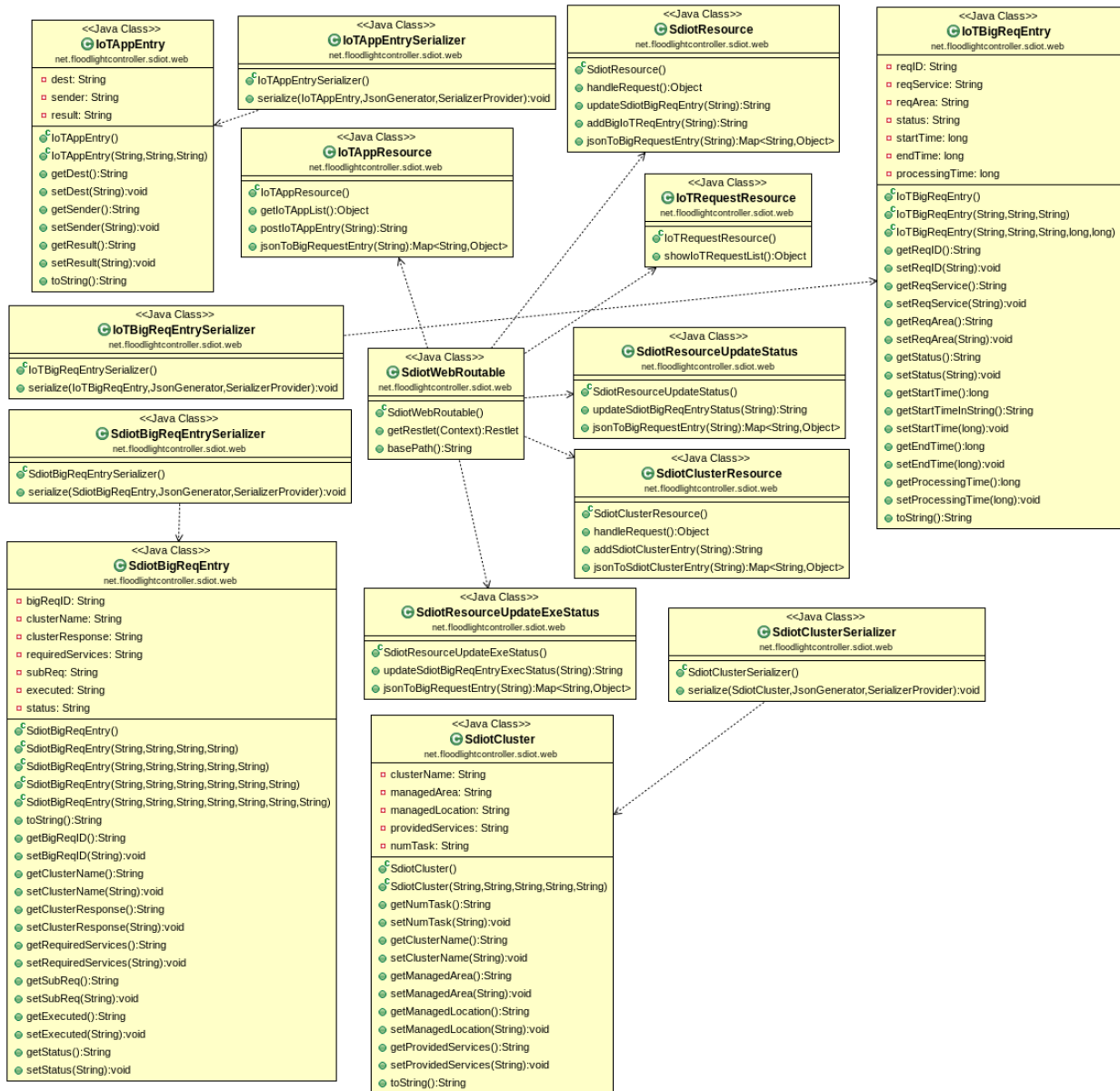


Figure 7.17 Class diagram for REST-API

❖ Data Storage

The SD-IoTC controller leverages collected information from the SDN controller core modules to achieve information about connected SD-IoT platforms. It builds its own data storage that is composed of three information tables. Table 1 stores information about all clusters' connection, such as IP addresses and MAC addresses that are used for flow configuration. The

table also keeps records about the “state” of each SD-IoT cluster. The “state” represents the number of tasks currently handled by the associated SD-IoT cluster. Table 2 and Table 3 store information regarding the capabilities of each SD-IoTC cluster. Details of the table 1,2,3 are shown in **Figure 7.18**, **Figure 7.19**, **Figure 7.20**, respectively.

```
mysql> select * from tab_iot_cluster;
+-----+-----+-----+-----+-----+-----+
| stt | location_id | cluster_name | IP_Address | Mac_Address | state |
+-----+-----+-----+-----+-----+-----+
| 1 | Area1 | sdiot01 | 10.0.0.1 | 00:00:00:00:00:01 | 10 |
| 2 | Area1 | sdiot02 | 10.0.0.2 | 00:00:00:00:00:02 | 10 |
| 3 | Area1 | sdiot03 | 10.0.0.3 | 00:00:00:00:00:03 | 0 |
| 4 | Area1 | sdiot04 | 10.0.0.4 | 00:00:00:00:00:04 | 30 |
| 5 | Area1 | sdiot05 | 10.0.0.5 | 00:00:00:00:00:05 | 50 |
+-----+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)
```

Figure 7.18 Table 1 - Connections to SD-IoT clusters

```
mysql> select * from tab_services;
+-----+-----+-----+-----+
| stt | service_id | service_name | service_description |
+-----+-----+-----+-----+
| 1 | SID01 | SER-NAME01 | Light |
| 2 | SID02 | SER-NAME02 | Ambient_Temperature |
| 3 | SID03 | SER-NAME03 | Pressure |
| 4 | SID04 | SER-NAME04 | Humidity |
| 5 | SID05 | SER-NAME05 | Infrared_Temperature |
+-----+-----+-----+-----+
```

Figure 7.19 Table 2 – Available IoT services

```
mysql> select * from tab_sdiot_services;
```

stt	location_id	cluster_name	service_id
1	Area1	sdiot01	SID01
2	Area1	sdiot01	SID03
3	Area1	sdiot01	SID05
4	Area1	sdiot02	SID02
5	Area1	sdiot02	SID03
6	Area1	sdiot02	SID04
7	Area1	sdiot03	SID01
8	Area1	sdiot03	SID02
9	Area1	sdiot03	SID03
10	Area1	sdiot03	SID04
11	Area1	sdiot03	SID05
12	Area1	sdiot04	SID01
13	Area1	sdiot04	SID02
14	Area1	sdiot04	SID03
15	Area1	sdiot04	SID04
16	Area1	sdiot05	SID02
17	Area1	sdiot05	SID03
18	Area1	sdiot05	SID04
19	Area1	sdiot05	SID05

Figure 7.20 Table 3 – SD-IoTC clusters’ capability

7.6 Performance Evaluation

In this section, we evaluate the proposed model and the implemented platform on two aspects: the implemented capability of the platform for provisioning large-scale IoT services on demand and the performance of the platform in terms of orchestration, coordination, configuration programming, and service provisioning.

7.6.1 Implementation Platform Capability

We will evaluate components and capabilities that contribute to the provisioning of large scale IoT services on demand. Three aspects are demonstrated: 1) Orchestration tasks and allocating of resources, 2) Configuration of resources to perform the allocated tasks, 3) Management of tasks, and the overall service. The capability is expressed by the following features:

At the cluster level, the SD-IoTC controller can:

- Update the capability of SD-IoT resources (**Figure 7.25**).

- Handle various requests demanding one or multiple IoT services (**Figure 7.26**).
- Handle one or multiple IoT requests at any time according to available SD-IoT resources (**Figure 7.26**).
- Orchestrate SD-IoT resources to process one or multiple IoT requests at any time (**Figure 7.28**).
- Re-orchestrate an IoT request if any allocated SD-IoT resource cannot handle an assigned request (**Figure 7.28a**).
- Program flows to transmit results to desired data collection points (**Figure 7.27**).
- Monitor the execution status of IoT service provision (**Figure 7.27**).

At the device level, the SD-IoTD controller can:

- Automatically update its available resources to the SD-IoTC controller (**Figure 7.25**).
- Update the capability of SDVSs as well as update the database about the status of the SDVSs (**Figure 7.29a**).
- Handle one/multiple requests at any time (**Figure 7.29c**).
- Process various types of requests for one or multiple services at any time (**Figure 7.29c**).
- Respond to the SD-IoTC controller if it cannot handle an assigned task (**Figure 7.29b**).
- Orchestrate SDVSs to achieve required IoT services and send obtained results to desired destinations (**Figure 7.30**).

❖ Interfaces for control and management

The programmability and orchestration of the LSSD-IoT model are demonstrated via the capability of orchestrating and configuring the underlying resources for provisioning IoT services on demand. The overall view of orchestration and programmability of the LSSD-IoT model happening at two levels is shown in **Figure 7.21**. The detailed results of programmability and orchestration at the two levels are collected and viewed as the template in **Figure 7.22**.

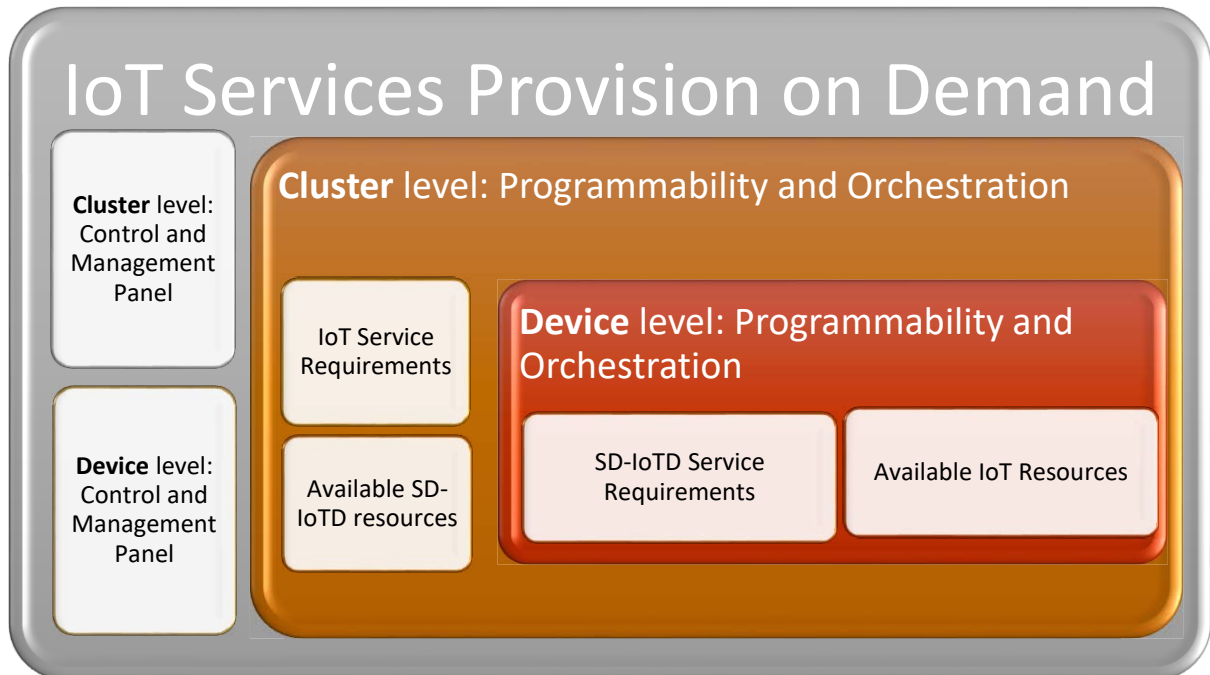


Figure 7.21 Level of programmability and orchestration of the LSSD-IoT platform

At the cluster level, there is one user interface for entering requests and four tables providing information about i) available SD-IoT resources, ii) requirements of IoT requests, iii) acknowledgment from IoT applications, and iv) results of resource orchestration and execution status.

At the device level, there is also one user interface for getting user demands or receiving requests from the SD-IoTC controller; and four tables presenting i) available SDVSs, ii) results of resource orchestration, iii) the number of messages exchanged between each SDVS and the SD-IoTD controller, and iv) SDVS's forwarding table, configuring table and sensor services.

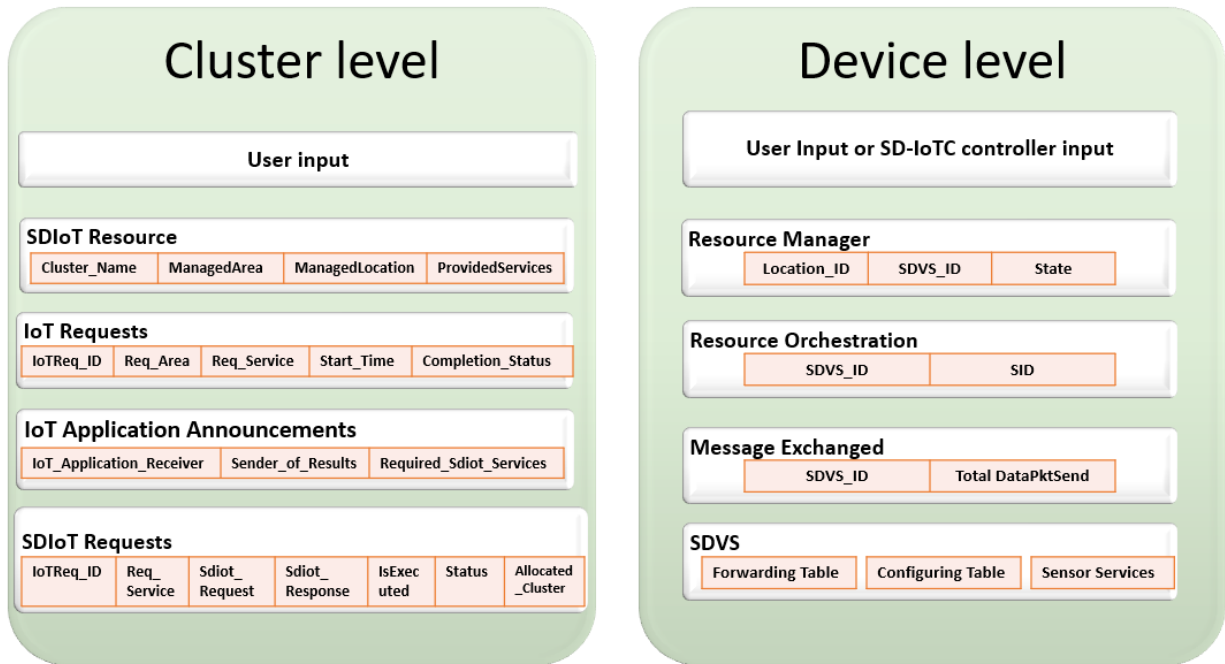


Figure 7.22 Control and management panel at the SD cluster and device level

We can access the SD-LoTC controller and put IoT requests via a web browser. Overall view of the orchestration and programmability results at the cluster and device levels are shown in **Figure 7.23** and **Figure 7.24**, respectively.

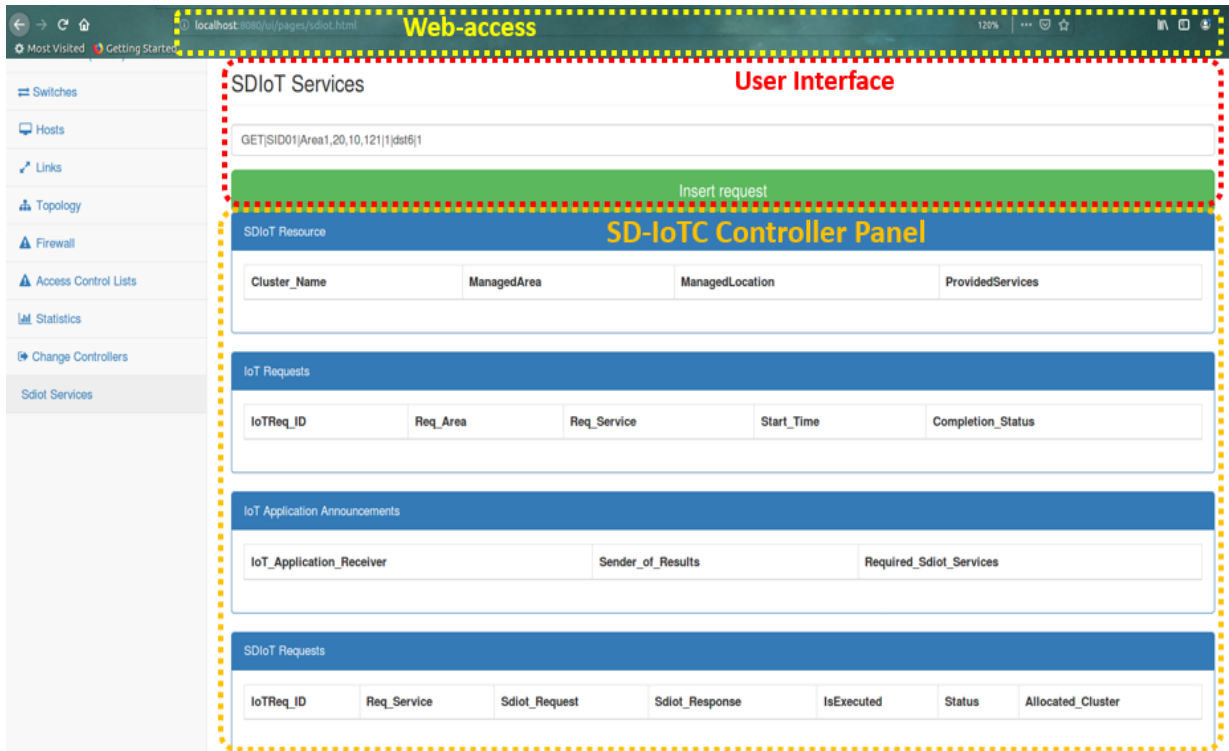


Figure 7.23 Overview of the control and management at the cluster level

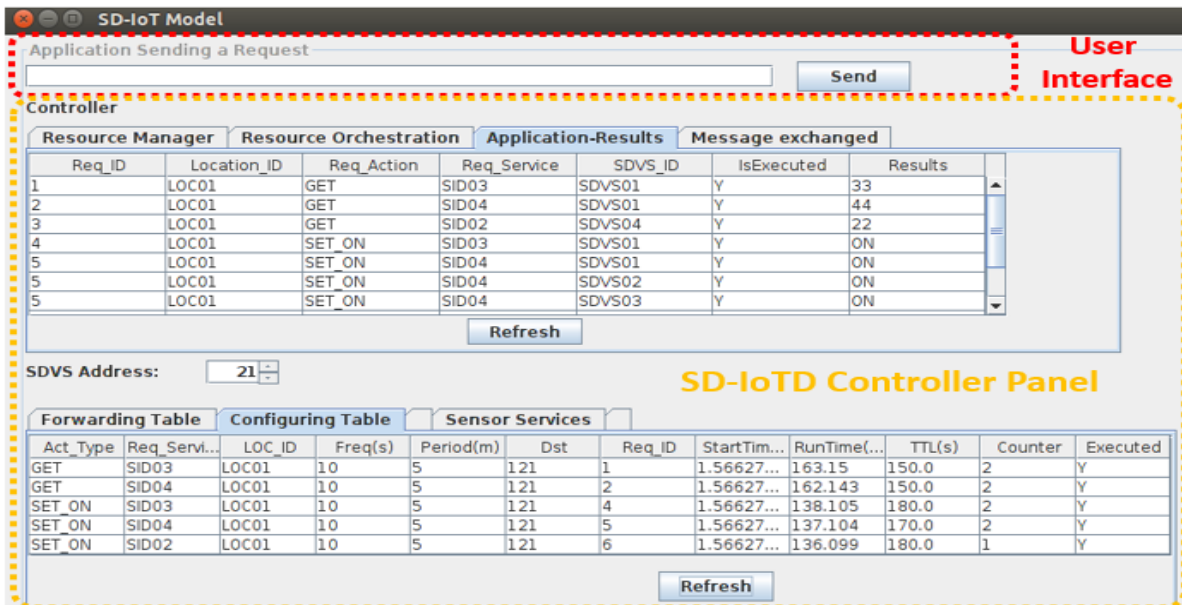


Figure 7.24 Overview of the control and management at the device level

❖ Orchestration and Scheduling – At the Cluster level

At this level, users are provided with an interface to specify their requirements, e.g., IoT services in which area, required period, how often results being sent to which destination (**Figure 7.23**). The users are also provided with available SD-IoT services (**Figure 7.25**). The resources are updated whenever an SD-IoTD cluster joins the LSSD-IoT system. In the area A1 (ManagedArea), there are currently five IoT clusters, e.g., sdiot01, sdiot02, with different capabilities (ProvidedServices). Each SD-IoTD cluster manages two locations (ManagedLocation), e.g., LOC01 and LOC02.

SDIoT Resource			
Cluster_Name	ManagedArea	ManagedLocation	ProvidedServices
sdiot01	Area1	LOC01,LOC02	SID01,SID03,SID05
sdiot02	Area1	LOC01,LOC02	SID02,SID03,SID04
sdiot03	Area1	LOC01,LOC02	SID01,SID02,SID03,SID04,SID05
sdiot04	Area1	LOC01,LOC02	SID01,SID02,SID03,SID04
sdiot05	Area1	LOC01,LOC02	SID02,SID03,SID04,SID05

Figure 7.25 Available SD-IoT resources

As shown in **Figure 7.26**, the LSSD-IoT can handle one or multiple requests that are marked with IoTReq_ID at any time (Start_Time). The input requests may require one or multiple IoT services (Req_Service). The user can monitor the status of IoT service achievement (Completion_Status) of required services if it is “OnGoing” or “Completely Done.” “OnGoing” indicates that the systems are still obtaining required services. “Completely Done” shows that all required services have been sent to desired destinations after a required period. For example, IoTReq_ID number 2 is with “Completely Done” status. As shown in **Figure 7.28b**, all sub-requests of request number 2 have been “Completely Done,” so the “Completion_Status” of the request is “Completely Done” (**Figure 7.26**). Meanwhile, regarding IoTReq_ID number 6, only

two out of five sub-requests have been done (**Figure 7.28b**), so its Completion_Status is still “OnGoing” (**Figure 7.26**).

IoT Requests				
IoTReq_ID	Req_Area	Req_Service	Start_Time	Completion_Status
1	Area1	SID01	1571550404154	Completely Done
2	Area1	SID04,SID02	1571550416288	Completely Done
3	Area1	SID03,SID04,SID05	1571550427916	OnGoing
4	Area1	SID03,SID04,SID05	1571550441393	OnGoing
5	Area1	SID01,SID02,SID03,SID04,SID05	1571550451705	OnGoing
6	Area1	SID05,SID03,SID04,SID01,SID02	1571550462736	OnGoing

Figure 7.26 IoT Requests Status

The LSSD-IoT system configures and monitors data flows between sources of IoT services (Sender_of_Results) and the destination of IoT results (IoT_Application_Receiver). Moreover, it also presents information on the achieved services (Required_Sdiot_Services). All the information is presented in **Figure 7.27**.

IoT Application Announcements		
IoT_Application_Receiver	Sender_of_Results	Required_Sdiot_Services
dest6	From sdiot01	results about serives SID03 for big request ID number 1
dest7	From sdiot04	results about serives SID01 for big request ID number 2
dest6	From sdiot02	results about serives SID04 for big request ID number 3

Figure 7.27 IoT Application Announcements

By default, SD-IoTD clusters can achieve all required services, but if any SD-IoTD cluster cannot handle a request, it sends a response (Sdiot_Response) regarding its incapability to the SD-IoTC controller. Hence, the SD-IoTC controller re-orchestrates other SD-IoTD resources to process the request (**Figure 7.28a**). As shown in **Figure 7.28a**, IoT request number 4 cannot be

handled by sdiot01, so sdiot05 is orchestrated to handle request number 4. Thus, after re-scheduling the resources, the SD-IoTC controller set the value for the Sdiot_Response to “New replacement of sdiot01 for IoT_Req_ID 4.”

SDIoT Requests						
IoTReq_ID	Req_Service	Sdiot_Request	Sdiot_Response	IsExecuted	Status	Allocated_Cluster
1	SID05	SET_ON SID05 LOC01,10,5,121 1 dst6 1	Can achieve all required services	YES	OnGoing	sdiot05
1	SID03	SET_ON SID03 LOC01,10,5,121 1 dst6 1	Can achieve all required services	YES	OnGoing	sdiot04
4	SID05	GET SID05 LOC01,10,5,121 1 dst6 4	New replacement of sdiot01 for IoT_Req_ID 4	YES	OnGoing	sdiot05
5	SID05	SET_OFF SID05 LOC01,10,7,121 1 dst7 5	Can achieve all required services	NO	OnGoing	sdiot01
5	SID03	SET_OFF SID03 LOC01,10,7,121 1 dst7 5	Can achieve all required services	YES	OnGoing	sdiot02
5	SID04	SET_OFF SID04 LOC01,10,7,121 1 dst7 5	Can achieve all required services	NO	OnGoing	sdiot02
5	SID02	SET_OFF SID02 LOC01,10,7,121 1 dst7 5	Can achieve all required services	NO	OnGoing	sdiot02
5	SID01	SET_OFF SID01 LOC01,10,7,121 1 dst7 5	New replacement of sdiot04 for IoT_Req_ID 5	NO	OnGoing	sdiot01

a) Reschedule results

SDIoT Requests						
IoTReq_ID	Req_Service	Sdiot_Request	Sdiot_Response	IsExecuted	Status	Allocated_Cluster
1	SID01	GET SID01 LOC01,20,2,121 1 dst6 1	Can achieve all required services	YES	Completely Done	sdiot01
2	SID04	SET_ON SID04 LOC01,10,2,121 1 dst7 2	Can achieve all required services	YES	Completely Done	sdiot02
2	SID02	SET_ON SID02 LOC01,10,2,121 1 dst7 2	Can achieve all required services	YES	Completely Done	sdiot02
3	SID03	SET_OFF SID03 LOC01,10,3,121 1 dst6 3	Can achieve all required services	YES	OnGoing	sdiot05
3	SID04	SET_OFF SID04 LOC01,10,3,121 1 dst6 3	Can achieve all required services	YES	OnGoing	sdiot05
6	SID05	GET SID05 LOC01,10,5,121 1 dst7 6	Can achieve all required services	YES	OnGoing	sdiot01
6	SID03	GET SID03 LOC01,10,5,121 1 dst7 6	Can achieve all required services	YES	OnGoing	sdiot01
6	SID04	GET SID04 LOC01,10,5,121 1 dst7 6	Can achieve all required services	YES	Completely Done	sdiot02
6	SID01	GET SID01 LOC01,10,5,121 1 dst7 6	Can achieve all required services	YES	OnGoing	sdiot01
6	SID02	GET SID02 LOC01,10,5,121 1 dst7 6	Can achieve all required services	YES	Completely Done	sdiot02

b) Status of orchestrated SD-IoTD clusters

Figure 7.28 SD-IoTC Controller – Resource Orchestration

The SD-IoTC controller can obtain the status of SD-IoTD resources allocated to an IoT request at a specific time (**Figure 7.28b**). For an IoT request (IoTReq_ID), a set of information associated with each IoT service (Req_Service) is kept: the responsible SD-IoTD cluster (Allocated_Cluster), the response status (Sdiot_Response), the execution status (IsExecuted), and the processing status (Status) which is “OnGoing” or “Completely done.” When an SD-IoTD cluster completes a task, it informs the SD-IoTC controller of its completion and marks the Status “Completely Done.”

❖ Orchestration and Scheduling – At the Device level

The control and management of each SD-IoTD cluster have been shown in the control panel of the SD-IoTD controller (**Figure 7.29, 7.30, 7.31**). The management panel includes three main windows presenting information collected from the application layer, the orchestration layer, and the device layer. The application window allows users to input their IoT requests or admit IoT requests from the SD-IoTC controller via a user interface (**Figure 7.24**).

The operation of the controller is shown through multiple tables, including i) Resource Manager, ii) Resource Orchestration, iii) Application-Results, iv) Message Exchanged.

Controller

a) Resource Manager

Location_ID	SDVS_ID	State
SDVS01	LOC01	9
SDVS02	LOC01	4
SDVS03	LOC01	4
SDVS04	LOC01	4
SDVS05	LOC02	4

b) Resource Orchestration

Application Sending a Request
GET|SID01|LOC01,20,10,121|1 Send

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

c) Application-Results

Req_ID	Location_ID	Req_Action	Req_Service	SDVS_ID	IsExecuted	Results
1	LOC01	GET	SID01	SDVS01	Y	11
1	LOC01	GET	SID02	SDVS01	Y	22
1	LOC01	GET	SID03	SDVS01	Y	33
1	LOC01	GET	SID04	SDVS01	Y	44
1	LOC01	GET	SID05	SDVS01	Y	55

d) Message exchanged

SDVS_ID	Total DataPktSend
SDVS01	51
SDVS02	11
SDVS03	11
SDVS04	11
SDVS05	11
SDVS06	11

Figure 7.29 SD-IoTD Controller – Resources Orchestration

The first table (**Figure 7.29a**) shows all SDVSs under the controller's management. For example, SDVS01 is in location LOC01 and has the "State" 9 that means the SDVS currently handles 9 tasks.

The second table (**Figure 7.29b**) displays appropriate SDVSs for a service request. For instance, for the incoming request, the SDVS01 is responsible for achieving services SID01,02,03,04,05.

The third table (**Figure 7.29c**) reveals the results with resources provided for application requests. From this table, we can see that the SDVS01 has achieved the required services (IsExecuted) and sent results (Results) to the required destination.

The last table (**Figure 7.29d**) displays the number of packets sent out by SDVSs. The SDVS01 has sent out a total of 51 data packets.

The device window is about the SDVS. It provides information regarding the Forwarding table, Configuring Table, and Sensor services provided by the SDVS (**Figure 7.30b**). The SDVS can be configured to provide desired services (**Figure 7.30c**) and forward data to the required destination (**Figure 7.30a**). The SDVS shows a list of capable services and their associated status. It also provides the details of each provisioned service (**Figure 7.30a**), including specific requirements such as the processing start time (StartTime), the processing duration (RunTime), and the remaining time (TTL). Moreover, it also shows the number of requests that are interested in a specific service. With such recorded information, the SD-IoTD controller can notify the SD-IoTC controller of the waiting time before the request can be processed.

SDVS

a) Forwarding table

SDVS Address: 21

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

Refresh Remove Modify

b) Sensor Services

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

Refresh Remove Modify

c) Configuring table

Act_T...	Req_...	LOC_ID	Freq(s)	Perio...	Dst	Req_ID	Start...	RunTi...	TTL(s)	Counter	Execu...
SET_ON	SID02	LOC02	30	5	121	0	1.545...	280.339	60.0	2	Y
GET	SID01	LOC01	20	10	121	1	1.545...	204.314	420.0	3	Y
GET	SID02	LOC01	20	10	121	1	1.545...	204.314	420.0	2	Y
GET	SID03	LOC01	20	10	121	1	1.545...	204.314	420.0	1	Y
GET	SID04	LOC01	20	10	121	1	1.545...	204.311	420.0	1	Y
GET	SID05	LOC01	20	10	121	1	1.545...	204.31	420.0	1	Y

Refresh

Figure 7.30 SD-IoTD Controller – Device Configuration

7.6.2 Platform Performance

In this section, we carry out the tasks of service provisioning and evaluate the performance through two performance measures: orchestration time and response time. Orchestration time is the time required for orchestrating a requested service at the cluster level. Response time is the service response time, which measures the time from when the SD-IoTC controller configures SD-IoT resources at the device level to the time the first lot of data is collected at the collection point. As illustrated in **Figure 7.31**, the Orchestration time is T1, the time needed for orchestrating SD-IoT clusters for serving an IoT request. Total Response time is composed of T2, T3, T4, T5, T61, and T62.

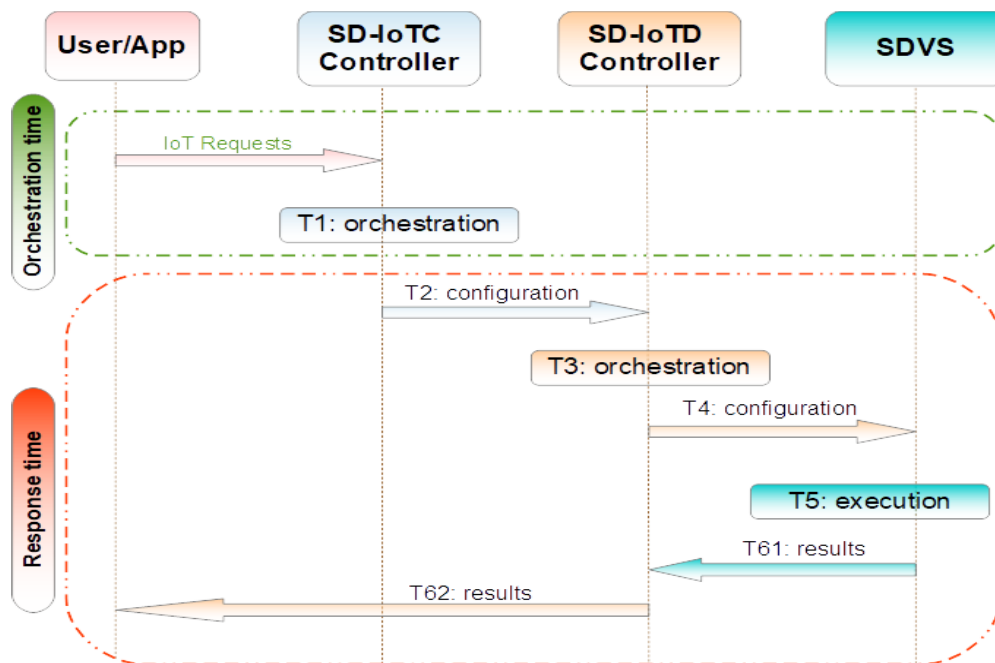
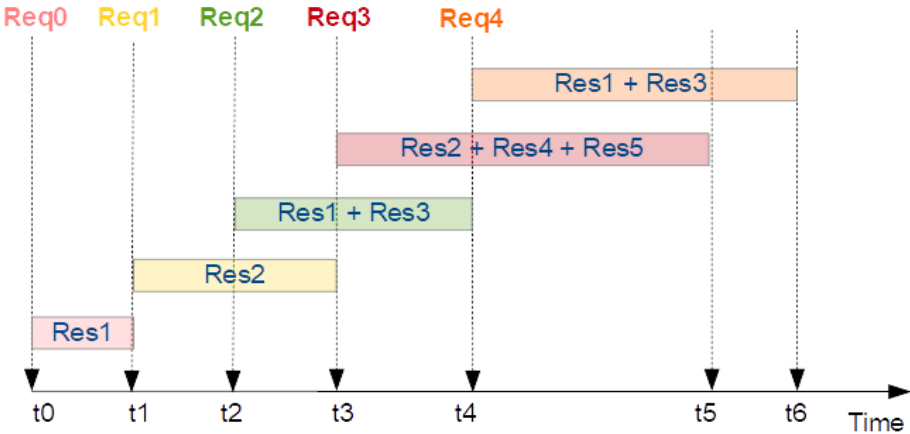


Figure 7.31 Timing diagram

The efficiency is examined via the above two parameters. The two measurements are investigated through two test cases: a) with optimization and b) non-optimization. With the optimized orchestration, the SD-IO TC controller reuses existing configurations/results associated with an IoT service to provision multiple IoT service demands that share similar interests. The LSSD-IoT system can save time in configuring SD-IoT resources to respond to multiple IoT requests as well as in the reduction of the load over the transport network since the model may only need to configure IoT clusters once to achieve services required by multiple requests. On the other hand, for the non-optimization case, the SD-IO TC controller needs to orchestrate available SD-IoT resources and configure them for every incoming request.

The orchestration is executed according to the availability of IoT resources and associated achieved IoT services, and current IoT requests. The SD-IO TC controller needs to know the amount of resources needed for handling current IoT requests, so it can allocate residual resources appropriately to incoming requests. The controller knows the number of current IoT requests and when they are processed, the allocated resources, and the needed duration. **Figure 7.32** shows the arrival of requests and the allocation of resources to the system at different times. At time t_0 ,

all system resources are available for allocation to meet Req0's demand. At time t1, Req0 has been served, and all the allocated resources to Req0 have been returned to the resource pool, and hence, the total system resources are again available for allocation to Req1's demand. However, at time t2, Req1 is still being served, and hence, the resources available for allocation to Req2 are the total system resources minus the amount of resources already allocated to Req1. Similarly, at an arrival time of a request, the orchestrator has to determine the amount of resources that have been allocated to existing requests and can only allocate residual resources from its resource pool to the new request.



Req_i: Requests arrived at the LSSD-IoT system at t_i (i=0,1,2...)
 Res_i : Resources and time needed for providing services to an IoT request Req_i
 Res1 Res2 Res3 Res4 Res5 Available resources

Figure 7.32 View of resource orchestration at the cluster level

We examine the orchestration for two cases a) with optimization (optimization case) and b) without optimization (non-optimization case). A number of simultaneous requests have been sent to the LSSD-IoT system. The requests may have one, two, three, or four common IoT services. As displayed in **Figure 7.33**, **7.34a**, and **7.34b**, the time needed for orchestrating and provisioning services for the two cases increases with the rising number of requests. Compared to the non-optimization case, the average time for orchestrating the same number of simultaneous requests

for the optimization case is longer due to the computation for reutilizing available IoT services that can be reused for provisioning IoT services to multiple IoT requests (**Figure 7.33**).

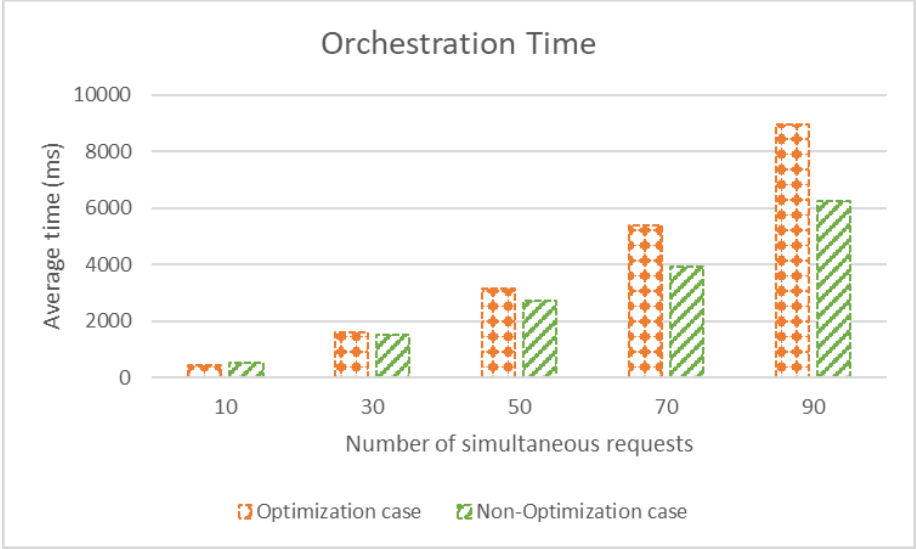
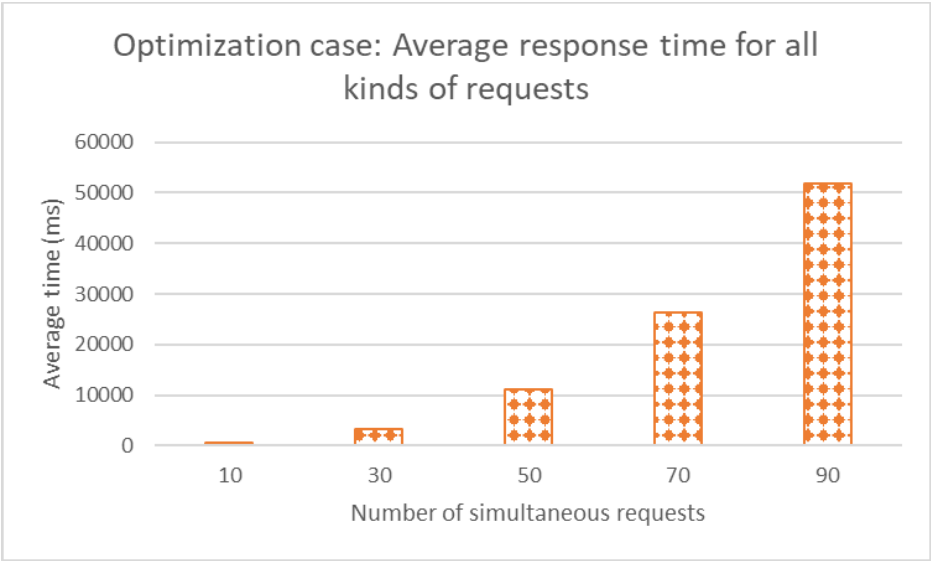
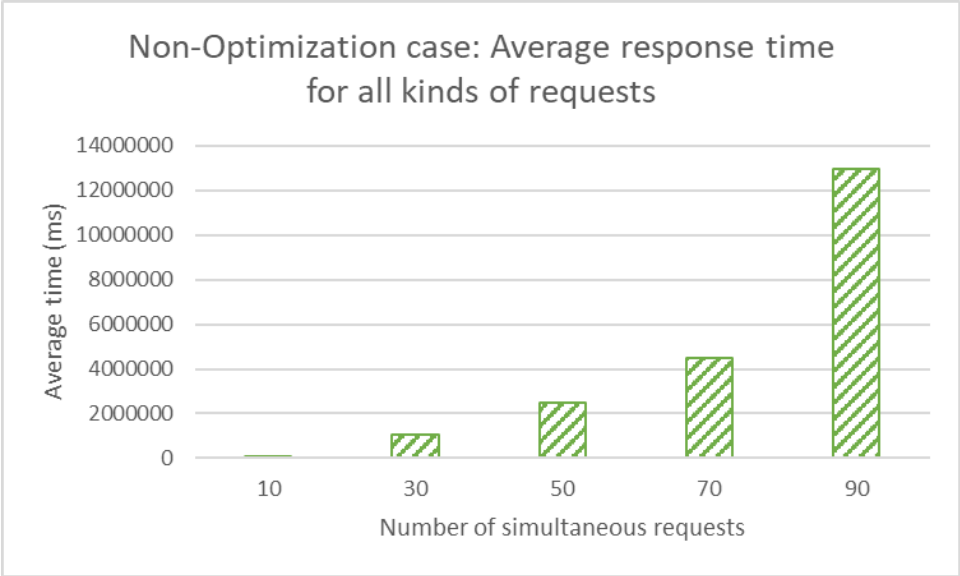


Figure 7.33 Orchestration time with and without optimization

In contrast, the response time for the non-optimization case is much longer than that of the optimization case, as depicted in **Figure 7.34a** and **Figure 7.34b** for provisioning IoT services to the same number of IoT requests. Particularly, for the optimization case, the maximum amount of time for responding to 90 concurrent requests is about 50000ms, while the non-optimization case requires about 249 times higher. This is because the optimization case a) reuses available results of IoT services and shares them with multiple IoT applications, and b) minimizes the configuration for the same type of services. The non-optimization case does not consider these factors, and hence orchestrating and configuring the underlying resources have to be performed for every incoming request.



a) Optimization case



b) Non-optimization case

Figure 7.34 SD-IoTC Controller – Response Time

To account for the increase in the orchestration time and response time for the optimization case, we investigate the time for 4 cases, as shown in **Figure 7.35**. A variety of requests are sent to the LSSD-IoT platform simultaneously. These requests may have a common interest in one, two, three, or four services. As presented in **Figure 7.35**, the more services required per request, the more time needed for orchestration as well as service response. In addition, for higher numbers of concurrent requests coming to the systems, the time for processing them becomes longer.



Figure 7.35 Orchestration and Response Time for optimization case with various input requests

7.7 Summary

In this chapter, we have introduced an LSSD-IoT model with new concepts to reshape the SDN and NFV technologies and overcome the limitations of IoT networked devices to support the programming of IoT services on demand. In particular, we attempt to incorporate SDN domain and SD-IoT domain for scalability in terms of the number of IT devices and their

geographical coverage for programming global IoT services on demand. We have developed and deployed a new SD-IoTC controller to enable it to control and manage both SDN devices and IoT devices. Details of the SD-IoTC controller's components are described. We discussed possible use cases of the proposed LSSD-IoT model and provided the operation of the model in the provision of IoT services on demand. We presented the feasibility and efficiency of the proposed model through the design and implementation of an LSSD-IoT platform prototype and evaluated the performance of the model.

Chapter 8 Conclusion and Future Work

This chapter summarizes this research and outlines the significant contributions. In association with the achievements, we then suggest future work.

8.1 Research Remarks

Internet of Things (IoT) applications have started becoming one of indispensable elements in various domains such as human life, social relationship, economy, education, health, industry, manufacturing, environment, and the government. However, the development is limited by a critical challenge of an increasing deployment cost and management of a rising number of sensors/IoT devices. In fact, the majority of IoT systems are rigid with little capability for programmable configuration or reusability as they are application-specific and manufacturer-specific. It is crucial to share IoT resources, including IoT devices, IoT infrastructure, and IoT services, among IoT applications in order to utilize IoT resources in an economical and timely manner.

In this research, we have identified a number of major issues of the current generation IoT systems/platforms. The first challenge is to manage the complexity of the interconnecting infrastructure of billions of IoT devices and harness their capabilities to serve not only local communities but also global communities in an efficient manner. The second challenging issue is to automate the provisioning of IoT services whenever they are needed on demand. The third challenge is in the developing of algorithms and supporting infrastructure for efficient utilization of the resources through sharing and reusing resources.

In order to address the challenging issues, we propose a large-scale software-defined IoT model and associated techniques for the provision of end-to-end services on demand.

In particular, on connectivity and networking architecture, Software-Defined Networking enables network programmability and fine-grained flow-based automated management that are not available with traditional distributed networks. Through the logically centralized knowledge of the whole network, an SDN controller can configure network devices automatically to deal with network dynamics. Many networks are currently being deployed for these purposes. Unfortunately, SDN-like programmability in IoT is still not being commonly used or is still being developed. We aim to adopt SDN for efficient deployment in the IoT domain and investigate a large-scale architecture spanning both the SDN domain and the IoT domain.

On programmable mechanisms for the orchestration of IoT services on demand, we attempt to enhance the capability of constrained IoT devices by using the virtual functions and the virtualized interface for orchestrating and programming services. For that purpose, we investigate algorithms and mechanisms for service orchestration.

As for communication and management protocol, sensors/IoT devices are not routing devices, and heavy protocols for programming network flows in network devices are not applicable to IoT devices. Efforts have been made to address this management issue with limited success. We investigate the development of a new simple protocol for adapting and enhancing the SDN paradigm to IoT networks to compensate for the different nature of network devices and IoT devices.

With regard to device capability, the highly resource-constrained nature of the IoT devices in terms of energy, computing power, storage, and wireless connectivity prevents a direct application of the wired SDN techniques to the IoT world. Many sensor/IoT devices with simple functionality do not possess a programmable interface which is required for resource and service sharing. Research efforts have been attempted to address this issue; systematically enriching devices with resource sharing capability remain open challenges. We investigate the virtualization technology to enhance and supplement the programmability of physical devices.

In addition, the IoT resources and services need to be shared and reused for developing multiple IoT applications, but the mechanism for that purpose has not been extensively explored. We investigate a programmable platform for sharing the underlying IoT resources and provisioning them on demand.

The contributions of this study can be summarized as follows.

We proposed a novel large-scale software-defined Internet of Things (LSSD-IoT) model that enables the provision of IoT services on demand by orchestrating IoT systems distributed over a large area. This model provides an end-to-end orchestration of IoT resources that involves orchestrating not only a single sensor/IoT device but also transporting infrastructure that connects geo-distributed IoT systems.

We introduced a software-defined virtual sensor (SDVS) that provides advanced capabilities for the constrained IoT devices. The SDVS allows the control and management of IoT devices and the network of these devices. It also enables the programmability of both the functional and forwarding behavior of the IoT devices in provisioning their capabilities to IoT applications. Through the SDVS, the IoT devices can be programmed for sharing their capabilities among IoT applications on demand.

We proposed a novel control and management protocol for programming IoT devices. The proposed protocol allows communication between IoT devices and the controller, which manages the device in sharing IoT resources with an external application. The protocol supports the controller in programming the IoT devices for provisioning IoT services on demand.

We proposed a software-defined IoT system that can be programmable and scalable to be a part of a large-scale IoT platform. The proposed architecture allows an IoT cluster to program its IoT resources, including IoT devices and the IoT network. In addition, via the SD-IoT model, the IoT cluster can be controlled, managed, and orchestrated by a global controller in provisioning IoT services on demand over a large-scale area.

Finally, we designed and implemented the proposed LSSD-IoT platform to demonstrate its capabilities and performance in the provision of IoT services on demand. The proposed platform

provides a new software-defined environment for integrating IoT and SDN domains as well as developing SDN-IoT-specific features.

On connectivity and networking architecture, we adapted SDN for efficient deployment in the IoT domain and demonstrated the use of the proposed LSSD-IoT architecture spanning both the SDN domain and IoT domain.

On device capability, we designed and implemented the Software-Defined Virtual Sensor to enhance and supplement the programmability of physical devices.

On communication and management protocol, we designed and developed S-MANAGE, a new simple protocol in a software-defined paradigm to manage IoT networks to compensate for the different nature of network devices and IoT devices.

On programmable mechanisms for orchestrating services on demand, we enriched the capability of IoT devices with virtual functions and interface for orchestrating and programming services. We developed algorithms and mechanisms for service orchestration.

On resources and services reusing and sharing, we investigated and implemented a programmable platform for sharing the underlying IoT resources and provisioning them on demand.

In summary, we believe that this thesis provides an affirmative answer to the posed question of “Can the SDN-NFV paradigm be leveraged for orchestration of geo-distributed IoT resources on resource-constrained IoT devices in provisioning IoT services on demand, and can the proposed model be realized in a practical implementation? Our proposal can be used to develop platforms for programmable IoT services of the future.

The novelty of this research lies in building a large-scale software-defined Internet of Things (LSSD-IoT) platform for provisioning IoT services on demand. The proposal includes a novel model of virtual sensors and a new LSSD-IoT architecture that is constructed from a cluster layer and a device layer. Each layer is implemented with a layer-specific software-defined controller and a new controller-sensor/cluster management protocol to provide a software platform for provisioning IoT services on demand.

The significance of this work is that it allows the orchestration and the programmability of IoT devices in the provision of IoT services on demand over a wide area. It enables i) IoT service providers to control end-to-end quality of services of IoT services provision over a large-scale IoT environment; ii) owners of IoT devices or IoT systems to be able to gain benefits from sharing their IoT resources; iii) IoT application developers to develop innovative and comprehensive IoT applications on demand with more options regarding QoS, security, mobility, or billing and without concern about the deployment of physical IoT infrastructure.

8.2 Future Work

In this study, we investigate technologies, network architectures, device capabilities, protocols, and programmable mechanisms for orchestrating IoT services on demand. The achievements open up a new research direction for end-to-end control and management of SDN-NFV-based IoT infrastructure in the provision of IoT services on demand. Although this research has significant research outcomes, there remain several limitations.

The proposed LSSD-IoT model can be a crucial platform for investigating new mechanisms associated with big data/QoS-driven network. Billions of devices produce a massive volume of data that causes a serious load on the transporting network. With the proposed LSSD-IoT model, we can configure data flow within a local IoT system and among transporting devices. This capability enables ease of developing and deploying a new paradigm in terms of QoS or big data management. In the future, we need to design QoS schemes to avoid conflicts among requirements from various IoT demands and improve the orchestration mechanism to maximize the response time in the provision of IoT services on demand.

In addition, security can be a major concern in a shared IoT platform. Therefore, for future research, we would study security policies that can cover both layers of the LSSD-IoT platform. At the cluster level, we need to establish trust between the LSSD-IoT platform and the SD-IoT platform when a local IoT system first joins a large-scale IoT platform. This protects both local and global IoT platforms. At the device level, we also need to ensure the each IoT device in an

IoT cluster is trusted for and is protected from sharing their IoT services with multiple IoT applications.

The proposed local SD-IoT model allows discrete IoT silos to share their resources with other IoT applications, and this brings benefits to both owners and users. Utilizing their local IoT system, the owners can achieve services for their own purposes, as well as they can obtain extra incomes by offering their resources to other users. This also allows users/developers to develop new IoT applications without concern about the deployment of a physical IoT infrastructure. For future work, we develop a service management scheme that enables the owner of each IoT system to manage their benefits from sharing their resources.

The proposed control and management S-MANAGE protocol is currently for control forwarding and functioning behavior of the SDVS in the provision of IoT services on demand. The protocol would be further developed to include more QoS-specific metrics but with a concern on the limitation of IoT networks and IoT devices.

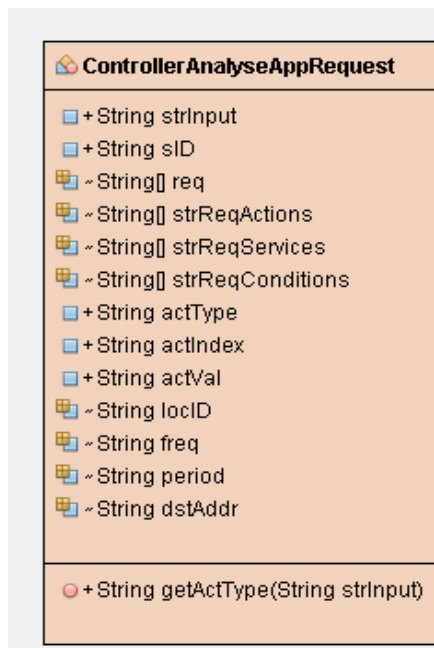
The proposed SDVS is proposed as an interface between the SD-IoT model and heterogeneous underlying IoT devices. Therefore, in the future, we would study and develop a library of drivers that allows the SDVS to communicate with heterogeneity of IoT entities. Besides that, we would investigate a discovery mechanism to allow mobile IoT devices to easily participate in a local IoT system to share their IoT values.

The proposed LSSD-IoT platform has been implemented and evaluated via a practical implementation. This makes the application of the SDN-NFV paradigm in an end-to-end control and management of IoT infrastructure for provisioning IoT services on demand become a reality. The efficiency of the proposed LSSD-IoT platform is evaluated by comparing two orchestration approaches deployed on the platform that uses the Floodlight SDN controller. The feature needs to be further investigated in various scenarios, including different SDN controller types and network sizes. Besides that, we would implement the LSSD-IoT platform with different types of IoT sensors/IoT devices and with different network sizes.

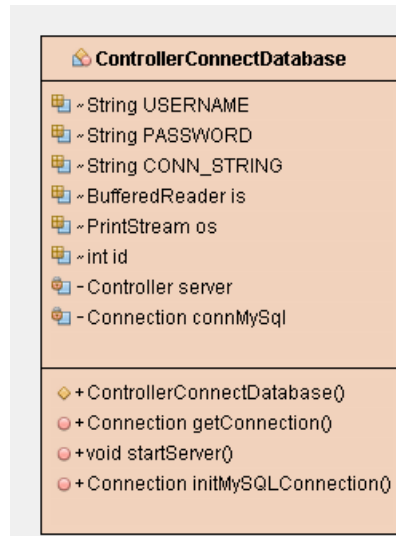
Appendices

Appendix 1 SD-LoTD controller – Software Components

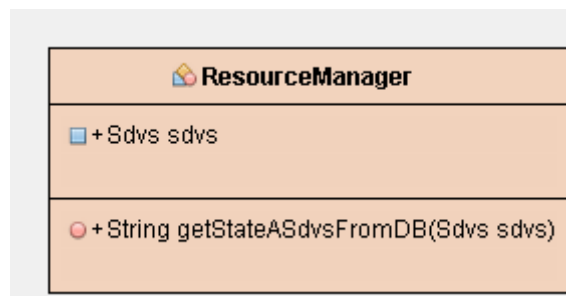
a) ControllerAnalyseAppRequest



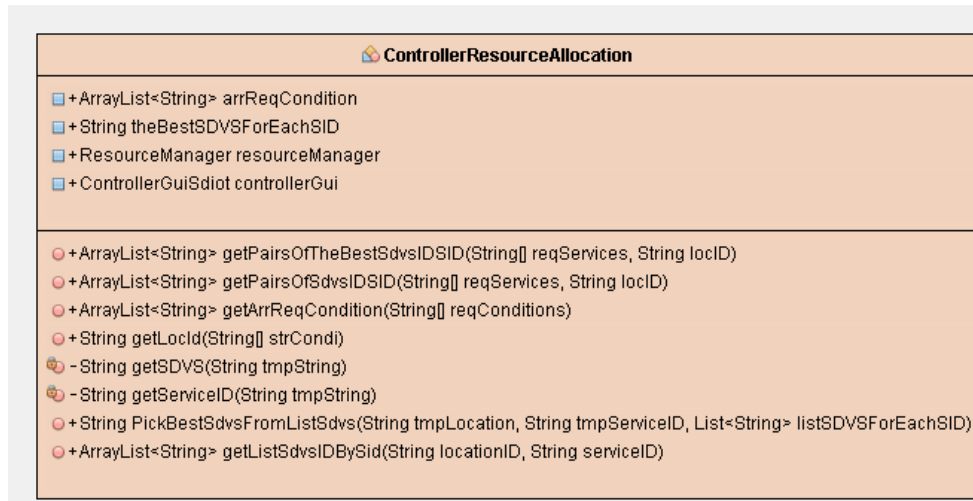
b) ControllerConnectDatabase



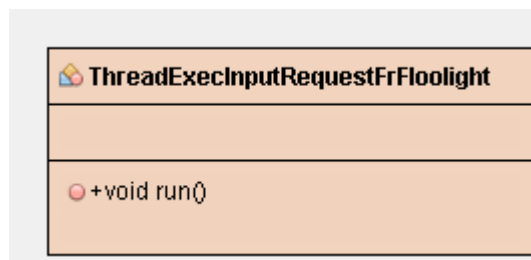
c) ResourceManager



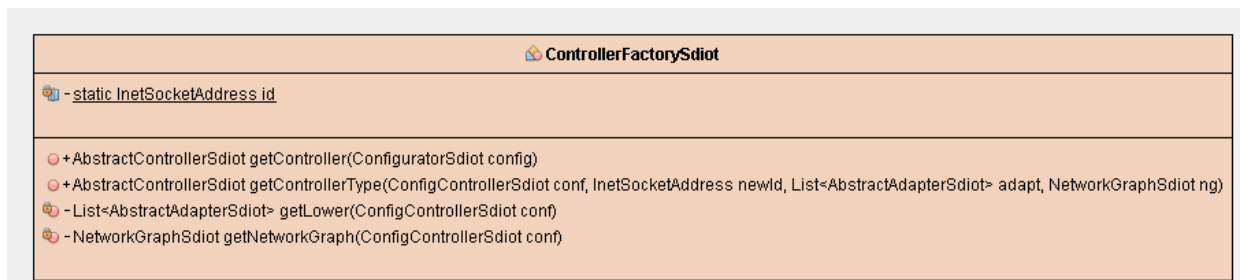
d) ControllerResourceAllocation



e) ThreadExecuInputRequestFloodlight



f) ControllerFactorySdiot




g) ControllerInterfaceSdiot



h) AbstractControllerSdiot

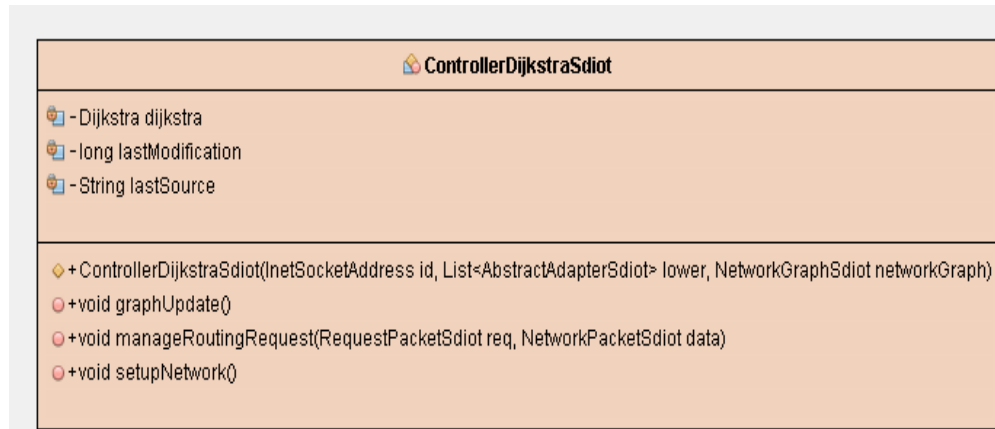
<<abstract>>

 **AbstractControllerSdiot**

-  -final int BUFF_SIZE
-  -final int DELAY
-  -final int FUNCTION_HEADER_LEN
-  -final int PARTS_MAX
-  -final int QUEUE_SIZE
-  #final int CACHE_EXP_TIME
-  #final Logger LOGGER
-  #final int RESPONSE_TIMEOUT
-  -ArrayBlockingQueue<NetworkPacketSdiot> bQ
-  -Map<String, ConfigPacketSdiot> configCache
-  -InetSocketAddress myId
-  -NetworkGraphSdiot networkGraph
-  -Map<String, RequestPacketSdiot> requestCache
-  -HashMap<NodeAddress, LinkedList<NodeAddress>> results
-  -NodeAddress sdvsAddress
-  +long startSendingResultToDst
-  +ArrayList<String> inRequests

-  -AbstractControllerSdiot(InetSocketAddress id, List<AbstractAdapterSdiot> lower, NetworkGraphSdiot network)
-  +static List<ConfigPacketSdiot> createConfigFunctionPackets(byte net, NodeAddress src, NodeAddress dst, byte id, byte[] buf)
-  +void addNodeAlias(byte net, NodeAddress dst, NodeAddress newAddr)
-  +void addNodeFunction(byte net, NodeAddress dst, byte id, String className)
-  +void addNodeRule(byte net, NodeAddress destination, FlowTableEntrySdiot rule)
-  +void addConfigRule(byte net, NodeAddress destination, ConfigRuleEntry rule)
-  +InetSocketAddress getId()
-  +NetworkGraphSdiot getNetworkGraph()
-  +NodeAddress getNodeAddress(byte net, NodeAddress dst)
-  +NodeAddress getControllerAddress()
-  +NodeAddress getNodeAlias(byte net, NodeAddress dst, byte index)
-  +List<NodeAddress> getNodeAliases(byte net, NodeAddress dst)
-  +int getNodeBeaconPeriod(byte net, NodeAddress dst)
-  +int getNodeEntryTtl(byte net, NodeAddress dst)
-  +int getNodeNet(byte net, NodeAddress dst)
-  +int getNodePacketTtl(byte net, NodeAddress dst)
-  +int getNodeReportPeriod(byte net, NodeAddress dst)
-  +int getNodeRssiMin(byte net, NodeAddress dst)
-  +FlowTableEntrySdiot getNodeRule(byte net, NodeAddress dst, int index)
-  +List<ConfigRuleEntrySdiot> getConfigRules(byte net, NodeAddress sdvsAddr)
-  +List<FlowTableEntrySdiot> getNodeRules(byte net, NodeAddress dst)
-  +HashMap<NodeAddress, LinkedList<NodeAddress>> getResults()
-  +NodeAddress getSdvsAddress()
-  +void managePacket(NetworkPacketSdiot data)
-  +long getStartTimeControllerSendResultToDst()
-  +void removeNodeAlias(byte net, NodeAddress dst, byte index)
-  +void removeNodeFunction(byte net, NodeAddress dst, byte index)
-  +void removeNodeRule(byte net, NodeAddress dst, byte index)
-  +void resetNode(byte net, NodeAddress dst)
-  +void sendPath(byte net, NodeAddress dst, List<NodeAddress> path)
-  +void setNodeAddress(byte net, NodeAddress dst, NodeAddress newAddress)
-  +void setNodeBeaconPeriod(byte net, NodeAddress dst, short period)
-  +void setNodeEntryTtl(byte net, NodeAddress dst, short period)
-  +void setNodeNet(byte net, NodeAddress dst, byte newNet)
-  +void setNodePacketTtl(byte net, NodeAddress dst, byte newTtl)
-  +void setNodeReportPeriod(byte net, NodeAddress dst, short period)
-  +void setNodeRssiMin(byte net, NodeAddress dst, byte newRssi)
-  +void setupLayer()
-  +void update(Observable o, Object arg)
-  -int getNodeValue(byte net, NodeAddress dst, ConfigProperty cfp)
-  -NetworkPacketSdiot putInRequestCache(RequestPacketSdiot rp)
-  -void register()
-  -ConfigPacketSdiot sendQuery(ConfigPacketSdiot cp)
-  +void sendNetworkPacket(NetworkPacketSdiot packet)
-  +void sendNetworkPacketSdiot(NetworkPacketSdiot packet)
- +void rxReqFrFloodlightViaSocket()
- +void rxReqFrFloodlightViaScanner()
- +ArrayList<String> getReqListFrFloodlight()

i) ControllerDijkstraSdiot



j) ControllerGuiSdiot

Appendix 2 SD-LoTD controller – Data Storage

```
mysql> select * from tab_location_sdvs;
```

stt	location_id	sdvs_id	service_id
1	LOC01	SDVS01	SID01
2	LOC01	SDVS01	SID02
3	LOC01	SDVS01	SID03
4	LOC01	SDVS01	SID04
5	LOC01	SDVS01	SID05
6	LOC01	SDVS02	SID01
7	LOC01	SDVS02	SID04
8	LOC01	SDVS02	SID05
9	LOC01	SDVS03	SID01
10	LOC01	SDVS03	SID02
11	LOC01	SDVS03	SID04
12	LOC01	SDVS03	SID05
13	LOC01	SDVS04	SID01
14	LOC01	SDVS04	SID02
17	LOC02	SDVS05	SID03
18	LOC02	SDVS05	SID05
19	LOC02	SDVS06	SID04
20	LOC02	SDVS06	SID05
21	LOC02	SDVS06	SID01
22	LOC02	SDVS06	SID03
23	LOC02	SDVS06	SID02
24	LOC02	SDVS07	SID03
25	LOC02	SDVS07	SID02
27	LOC02	SDVS08	SID05
28	LOC02	SDVS08	SID01
29	LOC02	SDVS08	SID02
30	LOC03	SDVS09	SID01
31	LOC03	SDVS09	SID04
32	LOC03	SDVS09	SID02
33	LOC03	SDVS09	SID05
34	LOC03	SDVS10	SID04
35	LOC03	SDVS10	SID03
36	LOC03	SDVS10	SID05
37	LOC03	SDVS11	SID01
38	LOC03	SDVS11	SID05
39	LOC03	SDVS12	SID03
40	LOC03	SDVS12	SID02
42	LOC01	SDVS02	SID02
43	LOC01	SDVS02	SID03
44	LOC01	SDVS03	SID03
45	LOC01	SDVS04	SID03
46	LOC01	SDVS04	SID04
47	LOC01	SDVS04	SID05
48	LOC02	SDVS05	SID02
49	LOC02	SDVS05	SID01
50	LOC02	SDVS05	SID04
51	LOC02	SDVS07	SID04
52	LOC02	SDVS07	SID01
53	LOC02	SDVS07	SID05
54	LOC02	SDVS08	SID03
55	LOC02	SDVS08	SID04

```
51 rows in set (0.02 sec)
```

```
mysql> select * from tab_sdvs_status;
```

stt	location_id	sdvs_id	state
1	LOC01	SDVS01	2
2	LOC01	SDVS02	2
3	LOC01	SDVS03	2
4	LOC01	SDVS04	2
13	LOC02	SDVS05	0
14	LOC02	SDVS06	0
15	LOC02	SDVS07	0
16	LOC02	SDVS08	0

```
mysql> select * from tab_services;
```

stt	service_id	service_name	service_description
1	SID01	SER-NAME01	Temperature
2	SID02	SER-NAME02	Touch
3	SID03	SER-NAME03	Proximity
4	SID04	SER-NAME04	Light
5	SID05	SER-NAME05	Movement

Bibliography

1. Columbus, L. *IoT Market Predicted To Double By 2021, Reaching \$520B*. 2018; Available from: <https://www.forbes.com/sites/louiscolumbus/2018/08/16/iot-market-predicted-to-double-by-2021-reaching-520b/#257cbfd41f94>.
2. Noura, M., M. Atiquzzaman, and M. Gaedke, *Interoperability in Internet of Things: Taxonomies and Open Challenges*. Mobile Networks and Applications, 2018.
3. Framingham, M. *The Growth in Connected IoT Devices Is Expected to Generate 79.4ZB of Data in 2025, According to a New IDC Forecast*. 2019 [cited 2019 12-November]; Available from: <https://www.idc.com/getdoc.jsp?containerId=prUS45213219>.
4. Ray, P.P., *A survey on Internet of Things architectures*. Journal of King Saud University - Computer and Information Sciences, 2018. **30**(3): p. 291-319.
5. Jones, N. *How to stop data centres from gobbling up the world's electricity*. 2018 [cited 2019 11 November]; Available from: <https://www.nature.com/articles/d41586-018-06610-y>.
6. Botta, A., et al., *Integration of Cloud computing and Internet of Things: A survey*. Future Generation Computer Systems, 2016. **56**: p. 684-700.
7. Mell, P.M. and T. Grance, *SP 800-145. The NIST Definition of Cloud Computing*. 2011, National Institute of Standards & Technology.
8. Yu, R., et al., *The Fog of Things Paradigm: Road toward On-Demand Internet of Things*. IEEE Communications Magazine, 2018. **56**(9): p. 48-54.
9. Mora, H., et al., *Collaborative Working Architecture for IoT-Based Applications*. Sensors (Basel, Switzerland), 2018. **18**(6): p. 1676.
10. Zhao, S., et al., *Internet of things service provisioning platform for cross-application cooperation*, in *Securing the Internet of Things: Concepts, Methodologies, Tools, and Applications*. 2020, IGI Global. p. 655-678.
11. Thoma, M., et al. *On IoT-services: Survey, Classification and Enterprise Integration*. in *2012 IEEE International Conference on Green Computing and Communications*. 2012.
12. Boulakbech, M., et al., *IoT Mashups: From IoT Big Data to IoT Big Service*, in *Proceedings of the International Conference on Future Networks and Distributed Systems*. 2017, ACM: Cambridge, United Kingdom.
13. Lee, D. and H. Lee, *IoT service classification and clustering for integration of IoT service platforms*. The Journal of Supercomputing, 2018. **74**(12): p. 6859-6875.
14. 5GPPP. *View on 5G Architecture*. 2019.
15. Gubbi, J., et al., *Internet of Things (IoT): A vision, architectural elements, and future directions*. Future Generation Computer Systems, 2013. **29**(7): p. 1645-1660.
16. El-Mougy, A., I. Al-Shiab, and M. Ibnkahla, *Scalable Personalized IoT Networks*. Proceedings of the IEEE, 2019. **107**(4): p. 695-710.
17. T.M.C Nguyen, D.H., *Large-scale Software-Defined IoTs Platform for Provisioning IoT Services on Demand*. International Journal of Smart Sensor Technologies and Applications (IJSSTA), 2020.

18. Nguyen, T.M.C. and D.B. Hoang. *Software-Defined Virtual Sensors for Provisioning IoT Services On Demand*. in *2020 International Conference on Information Technology and Internet of Things*. 2020. Shanghai, China: Scopus.
19. Nguyen, T.M.C. and D.B. Hoang. *S-MANAGE Protocol For Software-Defined IoT*. in *2018 28th International Telecommunication Networks and Applications Conference (ITNAC)*. 2018.
20. Nguyen, C. and D. Hoang, *S-MANAGE Protocol for Provisioning IoT Applications on Demand*. *Journal of Telecommunications and the Digital Economy*, 2019. **7**(3): p. 37-57.
21. Nguyen, T.M.C., D.B. Hoang, and Z. Chaczko. *Can SDN Technology Be Transported to Software-Defined WSN/IoT?* in *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)*. 2016.
22. Nguyen, T.M.C., D.B. Hoang, and T.D. Dang. *Toward a programmable software-defined IoT architecture for sensor service provision on demand*. in *2017 27th International Telecommunication Networks and Applications Conference (ITNAC)*. 2017.
23. 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.
24. Kothari, C.R., *Research methodology: Methods and techniques*. 2004: New Age International.
25. Ashton, K. *That 'Internet of Things' Thing : In the real world, things matter more than ideas*. 2009 [cited 2019 10 December]; Available from: <https://www.rfidjournal.com/articles/view?4986>.
26. Suresh, P., et al. *A state of the art review on the Internet of Things (IoT) history, technology and fields of deployment*. in *2014 International Conference on Science Engineering and Management Research (ICSEMR)*. 2014.
27. Marks, L.V. *Review: Nest Learning Thermostat*. 2013.
28. Hung LeHong , J.F. *Hype Cycle for Emerging Technologies, 2011*. 2011.
29. Atzori, L., A. Iera, and G. Morabito, *Understanding the Internet of Things: definition, potentials, and societal role of a fast evolving paradigm*. *Ad Hoc Networks*, 2017. **56**: p. 122-140.
30. *IEEE Towards a definition of the Internet of Things (IoT)*. 2015.
31. Bahga, A. and V. Madisetti, *Internet of Things: A Hands-On Approach*. 2014: Arshdeep Bahga & Vijay Madisetti.
32. Kafle, V.P., Y. Fukushima, and H. Harai, *Internet of things standardization in ITU and prospective networking technologies*. *IEEE Communications Magazine*, 2016. **54**(9): p. 43-49.
33. Haller, S., *The Things in the Internet of Things*. 2010.
34. Mohammed, F.H. and R. Esmail, *Survey on iot services: classifications and applications*. *Int J Sci Res*, 2015. **4**: p. 2124-7.
35. Y.2066, R.I.-T., *Common Requirements of the Internet of Things*. 2014.

36. Di Martino, B., et al., *Internet of things reference architectures, security and interoperability: A survey*. Internet of Things, 2018. **1-2**: p. 99-112.
37. Al-Fuqaha, A., et al., *Internet of things: A survey on enabling technologies, protocols, and applications*. IEEE Communications Surveys & Tutorials, 2015. **17**(4): p. 2347-2376.
38. Xiaojiang, X., W. Jianli, and L. Mingdong, *Services and key technologies of the internet of things*. ZTE Communications, 2010. **2**: p. 011.
39. Gigli, M. and S.G. Koo, *Internet of Things: Services and Applications Categorization*. Adv. Internet of Things, 2011. **1**(2): p. 27-31.
40. Chowdhury, A. and S.A. Raut, *A survey study on internet of things resource management*. Journal of Network and Computer Applications, 2018. **120**: p. 42-60.
41. Khan, R., et al. *Future Internet: The Internet of Things Architecture, Possible Applications and Key Challenges*. in *2012 10th International Conference on Frontiers of Information Technology*. 2012.
42. Yang, Z., et al. *Study and application on the architecture and key technologies for IOT*. in *2011 International Conference on Multimedia Technology*. 2011. IEEE.
43. Wu, M., et al. *Research on the architecture of Internet of Things*. in *2010 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE)*. 2010. IEEE.
44. Chaqfeh, M.A. and N. Mohamed. *Challenges in middleware solutions for the internet of things*. in *2012 international conference on collaboration technologies and systems (CTS)*. 2012. IEEE.
45. Sarwesh, P., N.S.V. Shet, and K. Chandrasekaran, *Envisioned Network Architectures for IoT Applications*, in *Cyber-Physical Systems: Architecture, Security and Application*, S. Guo and D. Zeng, Editors. 2019, Springer International Publishing: Cham. p. 3-17.
46. Nguyen, P., et al. *Advances in Deployment and Orchestration Approaches for IoT - A Systematic Review*. in *2019 IEEE International Congress on Internet of Things (ICIOT)*. 2019.
47. V, M., et al., *A Scalable Framework for Provisioning Large-Scale IoT Deployments*. ACM Trans. Internet Technol., 2016. **16**(2): p. 1-20.
48. Silva, d.C.J., et al., *Management Platforms and Protocols for Internet of Things: A Survey*. Sensors, 2019. **19**(3).
49. Noura, M., M. Atiquzzaman, and M. Gaedke, *Interoperability in Internet of Things: Taxonomies and Open Challenges*. Mobile Networks and Applications, 2019. **24**(3): p. 796-809.
50. Kreutz, D., et al., *Software-defined networking: a comprehensive survey*. Proceedings of the IEEE, 2015. **103**(1): p. 14-76.
51. Hoang, D., *Software defined networking—shaping up for the next disruptive step?* Australian Journal of Telecommunications and the Digital Economy, 2015. **3**(4).
52. Xie, J., et al., *Control plane of software defined networks: a survey*. 2015.
53. Stallings, W., *Foundations of modern networking: SDN, NFV, QoE, IoT, and cloud*. 2015: Addison-Wesley Professional.

54. Gupta, N.V.R. and M. Ramakrishna, *A Road Map for SDN-Open Flow Networks*. International Journal of Modern Communication Technologies & Research (IJMCTR), 2015. **3**(6).
55. Rowshanrad, S., et al., *A survey on SDN, the future of networking*. Journal of Advanced Computer Science & Technology, 2014. **3**(2): p. 232-248.
56. Wenfeng, X., et al., *A Survey on software-defined networking*. IEEE Communications Surveys & Tutorials, 2015. **17**(1): p. 27-51.
57. Lara, A., A. Kolasani, and B. Ramamurthy, *Network innovation using openflow: a survey*. Communications Surveys & Tutorials, IEEE, 2014. **16**(1): p. 493-512.
58. Black, C. and P. Goransson, *Software defined networks: a comprehensive approach*. 2014, Elsevier USA, Waltham, MA: Elsevier Science.
59. Nunes, B.A.A., et al., *A survey of software-defined networking: past, present, and future of programmable networks*. IEEE Communications Surveys & Tutorials, 2014. **16**(3): p. 1617-1634.
60. Fei, H., H. Qi, and B. Ke, *A survey on software-defined network and openflow: from concept to implementation*. IEEE Communications Surveys & Tutorials, 2014. **16**(4): p. 2181-2206.
61. Trevizan de Oliveira, B., M.C. Borges, and G.L. Batista. *TinySDN: Enabling multiple controllers for software-defined wireless sensor networks*. in *Communications (LATINCOM), 2014 IEEE Latin-America Conference on*. 2014.
62. Costanzo, S., et al. *Software defined wireless networks: unbridling sdn*. in *2012 European Workshop on Software Defined Networking (EWSDN)*. 2012. IEEE.
63. Gante, A.D., M. Aslan, and A. Matrawy. *Smart wireless sensor network management based on software-defined networking*. in *27th Biennial Symposium on Communications (QBSC)*. 2014. IEEE.
64. 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.
65. 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.
66. Galluccio, L., et al. *Reprogramming wireless sensor networks by using sdn-wise: a hands-on demo*. in *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. 2015. IEEE.
67. Mahmud, A. and R. Rahmani. *Exploitation of OpenFlow in wireless sensor networks*. in *2011 International Conference on Computer Science and Network Technology (ICCSNT)*. 2011.
68. Haleplidis, E., et al., *Software-defined networking (sdn): layers and architecture terminology*. 2015.
69. Kumar Somappa, A.A., K. Øvsthus, and L.M. Kristensen, *An industrial perspective on wireless sensor networks: a survey of requirements, protocols, and challenges*. IEEE Communications Surveys & Tutorials, 2014. **16**(3): p. 1391-1412.

70. Jayashree, P. and F. Infant Princy. *Leveraging sdn to conserve energy in wsn-an analysis*. in *3rd International Conference on Signal Processing, Communication and Networking (ICSCN)*. 2015. IEEE.
71. Yuan, A.S., H.-T. Fang, and Q. Wu. *OpenFlow based hybrid routing in Wireless Sensor Networks*. in *2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*. 2014. IEEE.
72. Han, Z.-j. and W. Ren, *A novel wireless sensor networks structure based on the SDN*. *International Journal of Distributed Sensor Networks*, 2014. **2014**: p. 1-7.
73. Kim, H. and N. Feamster, *Improving network management with software defined networking*. *IEEE Communications magazine*, 2013. **51**(2): p. 114-119.
74. Mahmud, A., R. Rahmani, and T. Kanter. *Deployment of flow-sensors in internet of things' virtualization via openflow*. in *Third FTRA International Conference on Mobile, Ubiquitous, and Intelligent Computing (MUSIC)*. 2012. IEEE.
75. Huang, R., et al., *Energy-efficient monitoring in software defined wireless sensor networks using reinforcement learning: a prototype*. *International Journal of Distributed Sensor Networks*, 2015. **2015**.
76. Costanzo, S., et al. *Software defined wireless networks: unbridling sdns*. in *European Workshop on Software Defined Networking (EWSDN)*. 2012. IEEE.
77. Hoang, D.B. and S. Farahmandian, *Security of Software-Defined Infrastructures with SDN, NFV, and Cloud Computing Technologies*, in *Guide to Security in SDN and NFV: Challenges, Opportunities, and Applications*, S.Y. Zhu, et al., Editors. 2017, Springer International Publishing: Cham. p. 3-32.
78. ETSI, *Network Functions Virtualisation (NFV) - Virtual Network Functions Architecture*. 2014.
79. Nitti, M., et al., *The Virtual Object as a Major Element of the Internet of Things: A Survey*. *IEEE Communications Surveys & Tutorials*, 2016. **18**(2): p. 1228-1240.
80. Madria, S., V. Kumar, and R. Dalvi, *Sensor Cloud: A Cloud of Virtual Sensors*. *IEEE Software*, 2014. **31**(2): p. 70-77.
81. Kabadayi, S., 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, IEEE Computer Society. p. 587-592.
82. Gupta, A. and N. Mukherjee. *Implementation of virtual sensors for building a sensor-cloud environment*. in *8th International Conference on Communication Systems and Networks (COMSNETS)*. 2016. IEEE.
83. Evensen, P. and H. Meling. *SenseWrap: A service oriented middleware with sensor virtualization and self-configuration*. in *2009 International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*. 2009.
84. Jordán Pascual Espada, et al., *Virtual Objects on the Internet of Things*. *International Journal of Artificial Intelligence and Interactive Multimedia*, 2011. **1**(4): p. 24-30.
85. Bröring, A., et al., *New Generation Sensor Web Enablement*. *Sensors*, 2011. **11**(3): p. 2652.
86. Omnes, N., et al. *A programmable and virtualized network & IT infrastructure for the internet of things: How can NFV & SDN help for facing the upcoming*

- challenges. in *2015 18th International Conference on Intelligence in Next Generation Networks*. 2015.
87. Mavromatis, A., et al. *A Software Defined Device Provisioning Framework Facilitating Scalability in Internet of Things*. in *2018 IEEE 5G World Forum (5GWF)*. 2018.
 88. Atzori, L., et al., *SDN&NFV contribution to IoT objects virtualization*. *Computer Networks*, 2019. **149**: p. 200-212.
 89. 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.
 90. Li, Y., et al. *A SDN-based architecture for horizontal Internet of Things services*. in *2016 IEEE International Conference on Communications (ICC)*. 2016.
 91. Qin, Z., et al. *A Software Defined Networking architecture for the Internet-of-Things*. in *2014 IEEE Network Operations and Management Symposium (NOMS)*. 2014.
 92. Farris, I., et al., *A survey on emerging SDN and NFV security mechanisms for IoT systems*. *IEEE Communications Surveys & Tutorials*, 2018: p. 1-1.
 93. Alenezi, M., K. Almustafa, and K.A. Meerja, *Cloud based SDN and NFV architectures for IoT infrastructure*. *Egyptian Informatics Journal*, 2018.
 94. Vilalta, R., et al. *End-to-end SDN orchestration of IoT services using an SDN/NFV-enabled edge node*. in *2016 Optical Fiber Communications Conference and Exhibition (OFC)*. 2016.
 95. Li, J., E. Altman, and C. Touati, *A General SDN-based IoT Framework with NVF Implementation*. *ZTE Communications*, 2015. **13**(3): p. 42-45.
 96. Jacobsson, M. and C. Orfanidis. *Using software-defined networking principles for wireless sensor networks*. in *11th Swedish National Computer Networking Workshop (SNCNW)*. 2015. Karlstad, Sweden.
 97. Anadiotis, A.-C.G., et al., *SD-WISE: a software-defined wireless sensor network*. arXiv preprint arXiv:1710.09147, 2017.
 98. Bera, S., et al., *Soft-WSN: Software-Defined WSN Management System for IoT Applications*. *IEEE Systems Journal*, 2016. **PP**(99): p. 1-8.
 99. Habibi, P., et al. *Virtualized SDN-Based End-to-End Reference Architecture for Fog Networking*. in *2018 32nd International Conference on Advanced Information Networking and Applications Workshops (WAINA)*. 2018.
 100. Muñoz, R., et al., *Integration of IoT, Transport SDN, and Edge/Cloud Computing for Dynamic Distribution of IoT Analytics and Efficient Use of Network Resources*. *Journal of Lightwave Technology*, 2018. **36**(7): p. 1420-1428.
 101. Sinh, D., et al. *SDN/NFV—A new approach of deploying network infrastructure for IoT*. in *2018 27th Wireless and Optical Communication Conference (WOCC)*. 2018. IEEE.
 102. K, K.O., et al. *Efficient Deployment of Service Function Chains (SFCs) in a Self-Organizing SDN-NFV Networking Architecture to Support IOT*. in *2018 Tenth International Conference on Ubiquitous and Future Networks (ICUFN)*. 2018.
 103. Anadiotis, A.G., et al., *Toward Unified Control of Networks of Switches and Sensors Through a Network Operating System*. *IEEE Internet of Things Journal*, 2018. **5**(2): p. 895-904.

104. Mouradian, C., N.T. Jahromi, and R.H. Glitho, *NFV and SDN-Based Distributed IoT Gateway for Large-Scale Disaster Management*. IEEE Internet of Things Journal, 2018. **5**(5): p. 4119-4131.
105. Tricomi, G., et al. *Software-Defined City Infrastructure: A Control Plane for Rewireable Smart Cities*. in *2019 IEEE International Conference on Smart Computing (SMARTCOMP)*. 2019.
106. ONF, *Use Cases for Carrier Grade SDN*. 2016.
107. Čolaković, A. and M. Hadžialić, *Internet of Things (IoT): A review of enabling technologies, challenges, and open research issues*. Computer Networks, 2018. **144**: p. 17-39.
108. Li, S., L.D. Xu, and S. Zhao, *5G Internet of Things: A survey*. Journal of Industrial Information Integration, 2018. **10**: p. 1-9.
109. 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.
110. Kortuem, G., et al., *Smart objects as building blocks for the Internet of things*. IEEE Internet Computing, 2010. **14**(1): p. 44-51.
111. Alshehri, A. and R. Sandhu. *Access Control Models for Cloud-Enabled Internet of Things: A Proposed Architecture and Research Agenda*. in *2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC)*. 2016.
112. Healy, M., T. Newe, and E. Lewis. *Wireless Sensor Node hardware: A review*. in *SENSORS, 2008 IEEE*. 2008.
113. Ray, P.P., *A survey on Internet of Things architectures*. Journal of King Saud University - Computer and Information Sciences, 2016.
114. INSTRUMENTS, T. *SimpleLink™ multi-standard CC2650 SensorTag™ kit reference design: TIDC-CC2650STK-SENSORTAG*. 2019 [cited 2019 01-December]; Available from: <http://www.ti.com/tool/TIDC-CC2650STK-SENSORTAG>.
115. Foundation, O.N., *OpenFlow Switch Specification, in Version 1.0.0 (Wire Protocol 0x01)*. 2009.
116. Foundation, O.N., *OpenFlow Configuration and Management Protocol OF-CONFIG 1.0*. 2011.
117. Milardo, S. *The stateful Software Defined Networking solution for the Internet of Things*. 2017; Available from: <https://github.com/sdnwiselab/sdn-wise-java>.
118. Deva Priya, I. and S. Silas. *A Survey on Research Challenges and Applications in Empowering the SDN-Based Internet of Things*. 2019. Singapore: Springer Singapore.
119. Sood, K., S. Yu, and Y. Xiang, *Software-Defined Wireless Networking Opportunities and Challenges for Internet-of-Things: A Review*. IEEE Internet of Things Journal, 2016. **3**(4): p. 453-463.
120. Razzaque, M.A., et al., *Middleware for Internet of Things: A Survey*. IEEE Internet of Things Journal, 2016. **3**(1): p. 70-95.
121. Asghari, P., A.M. Rahmani, and H.H.S. Javadi, *Internet of Things applications: A systematic review*. Computer Networks, 2019. **148**: p. 241-261.

122. Najjar, Y.S., *Gaseous pollutants formation and their harmful effects on health and environment*. Innovative energy policies, 2011. **1**: p. 1-9.
123. Alex, G., *IoT for smart cities: Use cases and implementation strategies*. 2018: ScienceSoft.