An IoT Gateway Middleware for interoperability in SDN Managed Internet of Things

By

Jyoti Budakoti

A thesis submitted to the Faculty of Graduate and Postdoctoral Affairs in partial fulfillment of the requirements for the degree of

Master of Applied Science

in

Electrical and Computer Engineering

Carleton University
Ottawa, Ontario

Abstract

Internet of Things (IoT) refers to interconnection of billions of things which include objects, services and living beings. The realization of IoT systems will fundamentally change how we interact with the world; a key technology in that direction is middleware. Middleware is an intermediary software system between IoT devices and application services. The objective of this thesis is to propose a lightweight middleware solution which can be deployed both on the Cloud (remote data centers) for intense analytics and on the Edge Network (nearby IoT Gateways) for local analytics to support near real time applications, and which supports interoperability between heterogenous devices and applications by providing multiple protocol bindings as plug and play services. A survey on existing middleware solutions is conducted with a thorough analysis of the challenges and the enabling technologies in developing an IoT middleware. The experiments are conducted on a SDN managed IoT network testbed and the results show that the proposed middleware solution is suitable for both Cloud (resourceful) and Edge network devices (IoT Gateway, which is designed on a resource constrained single board computer) and provides interoperability between IoT devices and applications.

Acknowledgements

I would like to express my gratitude to my academic supervisor Dr. Chung-Horng Lung for all his guidance and invaluable suggestions, as well as great discussions and providing me support in all possible ways to accomplish my research.

I would also like to thank Natural Sciences and Engineering Research Council of Canada (NSERC) and Ontario Centres of Excellence (OCE) for funding my thesis research.

I am grateful to my beloved family for their love and support which I felt despite the distance from home. Special thanks to Amit Singh Gaur for always being there to support and motivate me.

List of Abbreviations

API	Application Programming Interface
AMQP	Advanced Message Queuing Protocol
BLE	Bluetooth Low Energy
COAP	Constraint Application Protocol
CDB	Configuration Database
CRUD	Create Read Update Delete
CLI	Command Line Interface
CPU	Central Processing Unit
GPL	General Public License
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
ICMP	Internet Control Message Protocol
ID	Identifier
IoT	Internet of Things
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IOS	Internetwork Operating System
IP	Internet Protocol
JSON	JavaScript Object Notation
L2	Layer 2

L3	Layer 3
LAN	Local Area Network
M2M	Machine to Machine
NETCONF	Network Configuration Protocol
NMS	Network Management System
MQTT	Message Queuing Telemetry Transport
OID	Object Identifier
OS	Operating System
OUI	Organizationally Unique Identifier
RPC	Remote Procedure Call
RAM	Read Access Memory
RFID	Radio Frequency Identification
SBC	Single Board Computer
SDN	Software Defined Networking
SMI	Structure of Management Information
SNMP	Simple Network Management Protocol
SSH	Secure Shell
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TTL	Time to Live
UDP	User Datagram Protocol
URI	Unique Resource Identification

URL	Uniform Resource Locator
VM	Virtual Machine
VPN	Virtual Private Network
Web UI	Web User Interface
WLAN	wireless local area network
XML	Extensible Markup Language
XSD	XML Schema Definition

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The advancement in the field of automation, predominantly software, information of technology (IT) and network automation are bringing a paradigm shift in how the network will be shaped within a few years. A driving force in automation is from the huge application areas of Internet of things (IoT) which not only has the potential to impact how we work but also how we live. IoT is defined as a global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) objects based on existing and evolving interoperable information and communication technologies [1]. Around 11.2 billion connected objects are expected to be used worldwide in 2018, as per recent research forecasts in Gartner report [2]. This prediction outnumbers by 25 percent from 2017 and can reach up to 20 billion by 2020.

The largest user category of the connected objects is the consumer segment with around 5.2 billion units in 2017, as shown in Table 1.1. This category represents a total of about 63 percent of the overall number of applications in use.

Table 1.1 IoT Units Installed Based by Category (Millions of Units) [2]

Category	**2016**	**2017**	**2018**	**2020**
Consumer	3,963.0	5,244.3	7,036.3	12,863.0
Business: Cross-Industry	1,102.1	1,501.0	2,132.6	4,381.4
Business: Vertical-Specific	1,316.6	1,635.4	2,027.7	3,171.0
Grand Total	**6,381.8**	**8,380.6**	**11,196.6**	**20,415.4**

This rapid evolution of the IoT market has spurred an explosion in the number and variety of IoT solutions. As a result, the industries have shifted their focus in mass manufacturing and producing requirement specific hardware to cater these IoT solutions. Table 1.2 shows the figures for hardware spending from the Gartner Report in 2017 [2] and the use of connected objects among businesses is expected to drive $1,110 billion by 2018. Consumer applications is expected to be around $985 billion in 2018. By 2020, hardware spending from consumer and business segments can reach almost $3 trillion.

Table 1.2 IoT Endpoint Spending by Category (Millions of Dollars) [2]

Category	**2016**	**2017**	**2018**	**2020**
Consumer	532,515	725,696	985,348	1,494,466
Business: Cross-Industry	212,069	280,059	372,989	567,659
Business: Vertical-Specific	634,921	683,817	736,543	863,662
Grand Total	**1,379,505**	**1,689,572**	**2,094,881**	**2,925,787**

Today most of the manufacturers and providers are building their own IoT solutions with dedicated hardware devices, platforms and Cloud services with their propriety standards. With a lack of open standardization and inconsistency, it is becoming difficult or impossible to achieve interoperability, portability and manageability between different IoT devices and applications from different vendors.

The evolving IoT industries require a standard model in which IoT solutions can

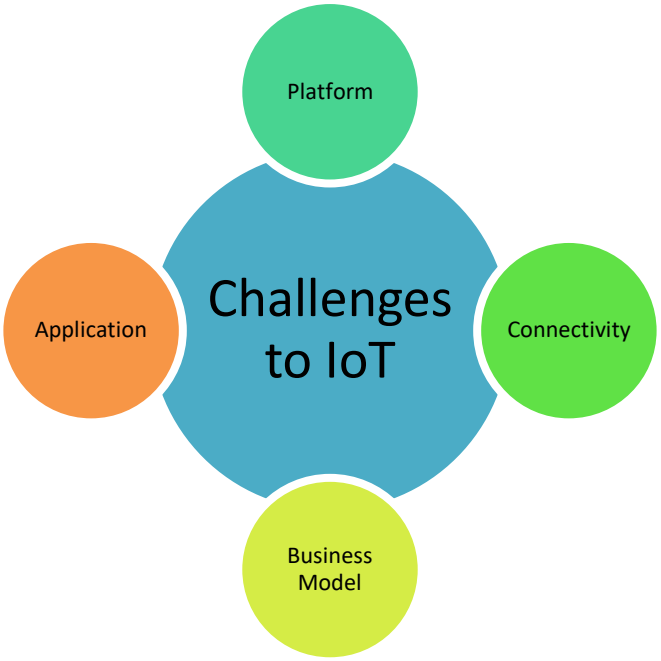

Figure 1.1 Challenges of IoT

interoperate with different devices and applications. Figure 1.1 depicts the major challenges to standardization of IoT, which are briefly explained below:

1) **IoT Platform:** This includes design of products, scalability in terms of wide adoption of protocols and analytics tools for handling massive volume of streaming data.
2) **Connectivity:** This phase includes adopting secure connectivity between devices, applications and humans. This field is largely dominated by machine to machine (M2M) communications.
3) **Business Model:** Designing a solid IoT model which satisfies all the requirements for all kinds of market.
4) **Applications:** IoT needs strong applications which can drive the business model using a unified platform. Applications must be able to control devices, collect and analyze data.

To overcome some of these challenges to meet interoperability between devices and application services, many researchers have proposed different Middleware solutions. Although a Middleware solution is an ideal platform to serve interoperability between Cloud applications and IoT devices, it is generally hosted on the Cloud and may introduce delay in near real-time applications [3]. One of such applications is autonomous driving, where vehicles equipped with sensors and self-driving technologies will be connected to other vehicles and the smart city infrastructure wirelessly. Autonomous driving will use many Cloud-based services to support in-car entertainment, predictive maintenance, remote diagnostics, etc. These Cloud based services can cater the need to maintain operational and maintenance logs. But the Cloud is not necessarily the best place for mission critical decision making that could help a vehicle to avoid a collision, given the time (latency) demands, this type of processing is best handled at the network Edge.

Edge computing enables IoT data to be processed closer to the source instead of sending it to a distant Cloud or data center. Edge and Fog Computing terms are often used interchangeably since both the concepts involve intelligence and processing being done closer to the source of data. In this thesis Fog and Edge computing are also used interchangeably.

Based on the above considerations, this thesis presents the design and implementation of an IoT Middleware solution for interoperability between devices transmitting data over different communication protocols like Wi-Fi, Bluetooth and Serial, and IoT applications built over protocols like HTTP, COAP, MQTT, and AMQP. The proposed Middleware solution is designed to be deployed at the Edge network, i.e., on the IoT Gateway, to support near real time local data analysis. Our proposed Middleware

solution acts as an IoT broker for Fog and Cloud application services. Fog application services provide fast computation and minimize delay for near real time streaming in comparison to Cloud.

1.1 Motivation

This thesis research is motivated to design and implement a distributed Middleware solution for IoT. A Middleware which acts as a broker for applications and can be deployed on both Edge and Cloud networks and support interoperability between IoT devices, platform and applications. A Middleware which also utilizes the SDN technology to support network management.

Compared to Cloud data centers, there are some distinctive characteristics of Fog or Edge computing [4] which makes Edge a better option for application services that require low latency or have limited bandwidth, some of which are listed below:

1) **Data close to user:** Fog computing allows data to be processed close to users instead of far data centers like in the Cloud. This minimizes the delay in data processing and transfer for latency intolerant applications.
2) **Great support for mobility:** There is a tremendous increase in the amount of data and devices. A Fog system supports the handling of large data transmissions and provides a better and faster way to access and analyze the data.
3) **Reducing network traffic:** Fog computing is a great option to prevent inappropriate or irrelative information to travel to the overall network. Only aggregated or periodic data could be transmitted to the Cloud or a data centers. By only sending important data

over the network, the Fog or Edge computing system reduces the data traversing the network.

4) **Support for IoT:** Fog computing can be applied in various application areas of IoT, e.g., smart traffic lights and vehicles, smart grids, smart cities, wireless sensor network and actuator and cyber-physical systems.

1.2 Contributions

The main contributions of this thesis can be summarized as below:

1) This thesis presents a proposed Middleware solution as an IoT broker which can effectively handle IoT data from multi-protocol enabled devices and transmit data to applications working on different data protocols, thus achieving interoperability between heterogenous devices, applications and protocols.
2) This thesis also presents a distributed Middleware approach as an efficient solution to a hybrid Fog and Cloud network to cater low latency applications and deep analytics enterprise applications simultaneously. This is achieved by deploying the proposed Middleware solution as an IoT broker on a IoT gateway (Edge) and the Microsoft Azure Cloud.
3) This thesis explores the potential gain of deploying Middleware close to the Edge network for Fog computing. The experimental results obtained by deploying the Middleware on the Edge (a nearby IoT Gateway) demonstrates that our proposed Middleware is a lightweight solution and can efficiently work on resource constrained IoT Gateways (Raspberry Pi 3 in this thesis)

4) This thesis also presents an architecture design of how IoT networks can be managed through software-defined networking (SDN) technologies.

1.3 Methodology

The focus of this thesis is to design and implement a lightweight Middleware solution which can be deployed on the Edge Network (a nearby IoT Gateway) as well as on the Cloud and support interoperability between heterogenous IoT devices and applications.

The first phase of this thesis involved an investigation and selection of the required hardware devices like sensors, Gateway, SDN switches, and SDN controller to build SDN based IoT network testbed for experiments. A significant part of this thesis is devoted to the analysis of the challenges faced in the current IoT solutions and the design of a unified data integration approach for applications interoperability.

The next stage included building a SDN managed IoT network testbed using hardware devices and connecting the SDN-based IoT network with the Microsoft Azure Cloud network for further experiments. The key component in this thesis is the design of a Middleware IoT solution which acts an IoT broker and deploying it on Edge network (nearby IoT gateway) and on Microsoft Azure Cloud.

1.4 Thesis Outline

This chapter serves as an introduction to the entire thesis. A brief synopsis of the remaining chapters are as follows.

Chapter 2 discusses relevant background information on Network Softwarization technologies like SDN, Virtualization, Cloud, IoT which are helpful in designing a

Middleware solution. The Chapter also discusses the results of our literature study on the existing Middleware design.

Chapter 3 focuses on the detailed implementation of the SDN managed IoT network architecture. The chapter presents design and implementation of the proposed Middleware solution and describes Device Discovery Module algorithms for auto device discovery in Wi-Fi, serial and Bluetooth network.

Chapter 4 presents experiment results and performance analysis of the proposed Middleware solution in terms of deployment at Edge network and Cloud for devices and applications interoperability.

Chapter 5 concludes the thesis and summaries the objectives achieved and suggests future research directions.

Chapter 2

Background and Related Work

This chapter introduces the underlying concepts and related work in management challenges and Middleware solution for IoT. This chapter also focuses on key technologies driving design and deployment of Middleware solutions. Section 2.1 introduces to the concept of Network Softwarization and its characteristics as new paradigm to network design and management. Section 2.2 provides the introduction to Network Softwarization in IoT applications and SDN. Section 2.3 discusses about management challenges in IoT networks and Section 2.4 discusses about IoT Middleware as a solution to some of the challenges and requirements for Middleware design. Section 2.5 discusses key challenges and issues in designing a IoT Middleware solution. Section 2.6 presents literature review in the relevant areas and discusses how the new approach presented in this thesis is efficient in comparison with related work. The chapter is summarized in Section 2.7.

2.1 Introduction to Network Softwarization

Network Softwarization term was first introduced at the IEEE conference NetSoft in 2015 [5]. Network Softwarization has greater emphasis on emerging network technologies like IoT, Software Defined Networking, Virtualization and Edge & Cloud Computing. It is focused on utilizing software for designing, deploying, maintaining network components and optimizing network and services architectures [5].

2.1.1 Characteristics of Network Softwarization

FG IMT-2020 report by Internet Engineering Task Force (IETF), the organization to develop and promote internet standards, illustrates that Network Softwarization helps to improve global system qualities like usability, modifiability, effectiveness, security and efficiency, testability, maintainability, reusability, extensibility, portability and scalability [6]. Network Softwarization supports greater flexibilities in the network by promoting levels of abstraction and programmability in application services. The major components of Network Softwarization are discussed in detail in the following sections.

2.1.2 Network Function Virtualization (NFV)

NFV [7] is the virtualization of network functions as a software component which can be deployed on hardware resources. NFV promotes Softwarization by minimizing dependency on the dedicated network hardware devices. Some of the advantages of using NFV are presented as follows:

1) **Lower CapEx/OpEx**: Minimizing the need to purchase dedicated hardware devices and reducing resource requirements like space, power and cooling. Provide management of network services.
2) **Accelerate Deployment**: As NFV is a software component, hence it can be designed as per user requirement and deployed quickly to meet the needs.
3) **Deliver Agility and Flexibility**: NFV supports agility and flexibility by changing services to adapt to changing business demand.

To standardized, develop requirements and architecture for virtualizing network

function in telecom networks a group was formed, known as The ETSI Industry Specification Group for Network Functions Virtualization (ETSI ISG NFV) [8]. Figure 2.1 presents NFV architectural framework by ETSI, called NFV MANO. NFV MANO is a framework for NFV management and orchestration.

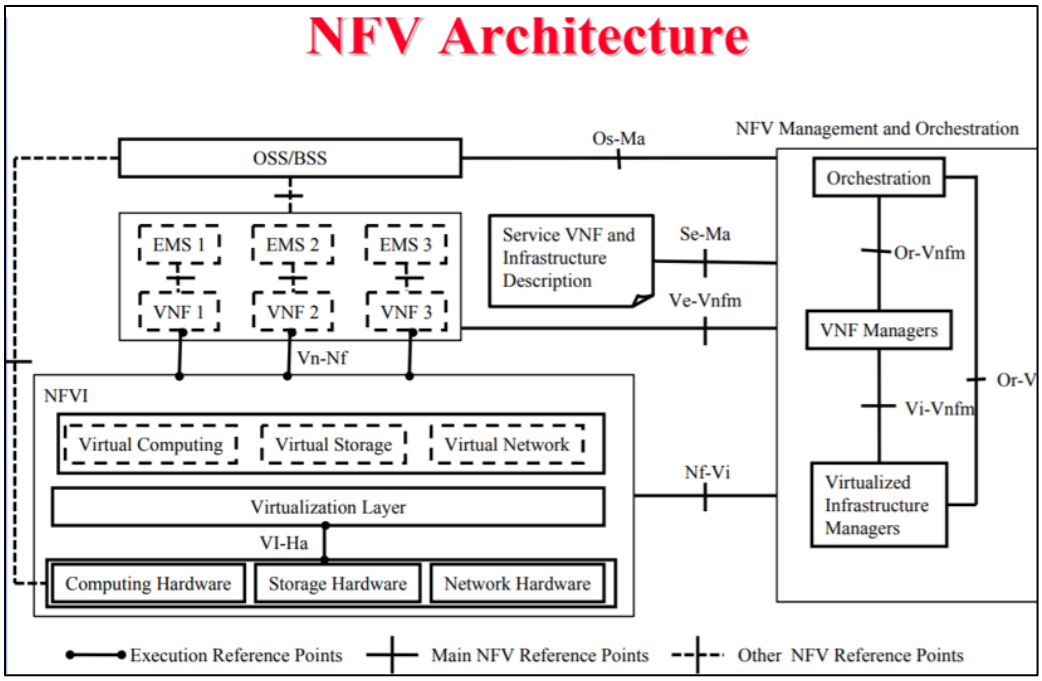

Figure 2.1 NFV Architectural framework by ETSI [8]

2.1.3 Software-Defined Networking (SDN)

SDN [9] decouples network control from the forwarding plane and provide a centralized management of the network resources using software applications. It abstracts the underlying network infrastructure from the application services, which provide direct programmability of the network control. The foundation of a SDN solution is OpenFlow protocol [10]. A simplified SDN architecture is presented in Figure 2.2. The Infrastructure layer is abstracted from the Application layer. Applications communicate with the Control

layer via Application Programming Interface (API). The SDN Control layer manages the network devices at the Infrastructure layer by communicating over OpenFlow protocol.

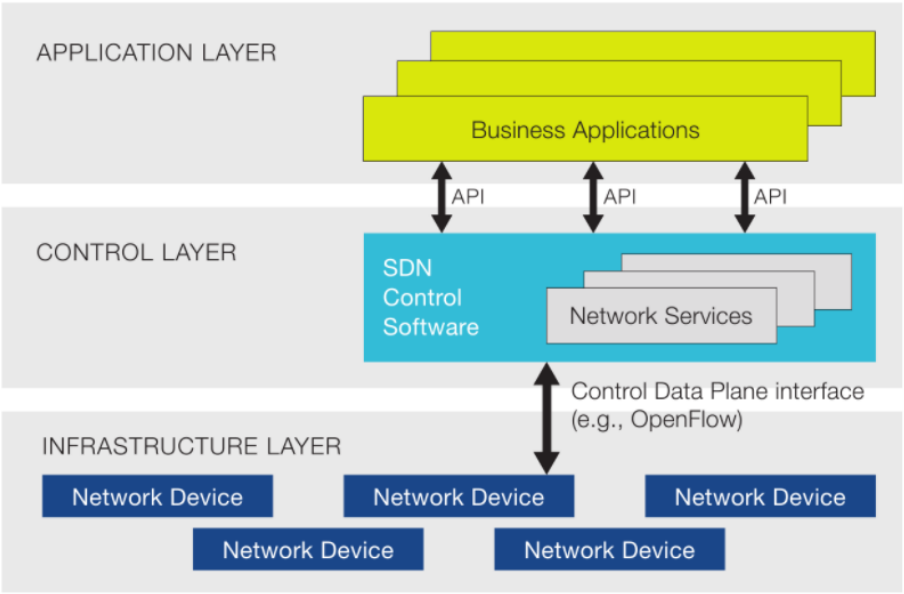

Figure 2.2 SDN Architecture [11]

The following are the characteristics of a SDN architecture:

1) **Centralized Management:** The network intelligence is centralized at Control layer, which controls and manages the network devices at Infrastructure layer.
2) **Direct Programmability:** SDN offers direct programmability as it decouples the control from the forwarding plane.
3) **Support for Open Standard:** SDN supports OpenFlow, which is an open standard protocol.
4) **Support Network Virtualization**: SDN can play an important role in the orchestration of the NFV Infrastructure resources, both physical and virtual, enabling functionality such as provisioning, configuration of network connectivity, bandwidth allocation,

automation of operations, monitoring, security, and policy control. SDN can provide the network virtualization required to support multi-tenant NFVIs. [12]

2.1.4 Software-Defined Network and Cloud

Cloudification enables ubiquitous access to shared services, shared pool of computing and storage resources, and connectivity on cloud network. It provides users and providers with various capabilities to process and store their data and services in data centers. It relies on sharing of resources to achieve coherence and uses virtualization techniques such as abstraction, pooling, and automation to all of the connectivity, compute and storage to achieve network services. It could take also the kind of mobile edge computing architecture where Cloud Computing capabilities and an IT service environment are available at the edge of the mobile network or Fog architecture that uses one or a collaborative multitude of end-user clients or near-user edge devices to execute a substantial amount of services (rather than in Cloud data centers), communication (rather than routed over the Internet backbone), and control, configuration, measurement and management.

2.1.5 Fog Computing

The term “Fog Computing” was coined by Cisco [13] which refers to the concept of extending cloud computing closer to the edge or users [14]. Fog computing facilitates compute, storage and network operations between Cloud data centers and end user devices. Fog computing can provide better quality of service (QoS) in terms of reduced energy consumption, reduced latency and reduced data traffic over the internet, etc. [15].

An architectural diagram for comparison in Fog and Cloud is presented in Figure 2.3 and a tabular comparison in Table 2.1

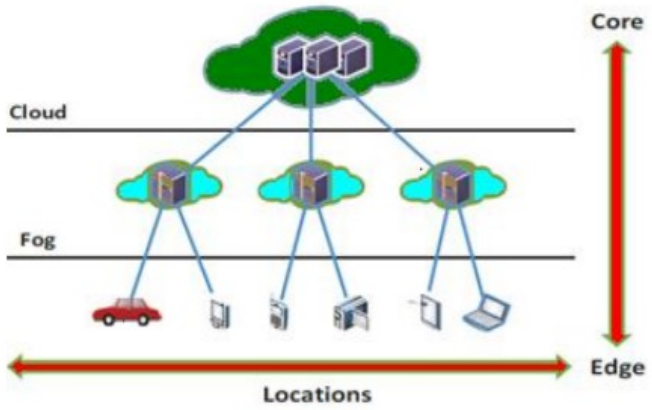

Figure 2.3 Cloud and Fog comparison on Edge [15]

Table 2.1 Comparisons of Cloud and Fog on Different parameters [16]

Parameters	**Cloud Computing**	**Fog Computing**
Server nodes location	Within the Internet	At the edge of the local network
client and server distance	Multiple hops	Single hop
Latency	High	Low
Delay Jitter	High	Very low
Security	Less secure, Undefined	More secure, Can be defined
Awareness about location	No	Yes
vulnerability	High probability	Very low probability
Geographical distribution	Centralized	Dense and Distributed
Number of server nodes	Few	Very large
Real time interactions	Supported	Supported
kind of last mile connectivity	Leased line	Wireless
Mobility	Limited support	Supported

2.2 Network Softwarization in IoT

IoT can be viewed as a global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) objects based on existing and evolving interoperable information and communication technologies (ICTs) [17]. IoT protocols and the communication stack of IoT is shown in Figure 2.4.

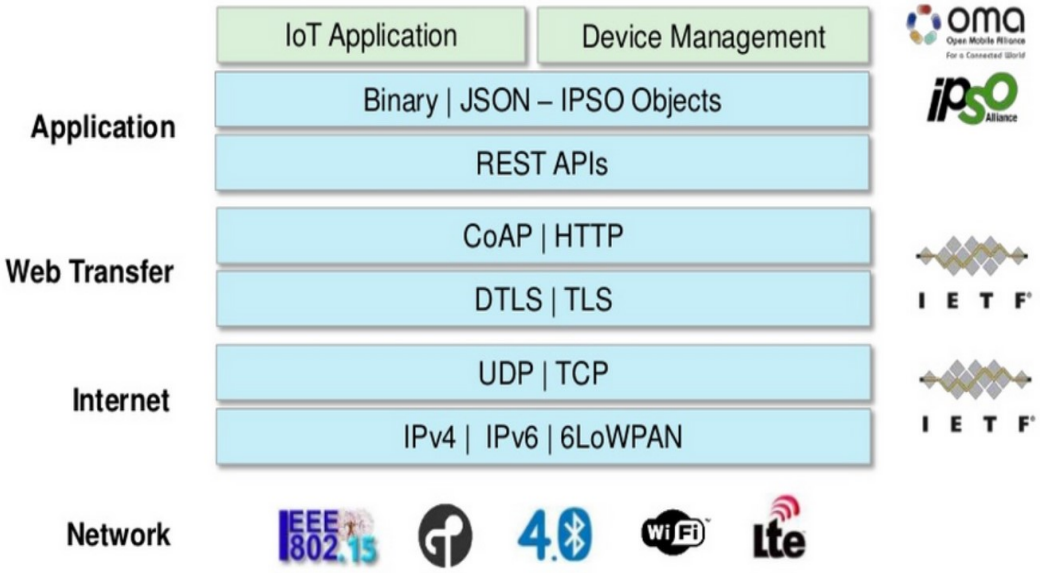

Figure 2.4 IoT communication stack [18]

2.2.1 IoT Messaging protocols

This section discusses about the popular emerging messaging IoT protocols. There are many protocols but, four IoT protocols are widely accepted in IoT systems for messaging. These four popular application protocols in IoT are MQTT, HTTP, CoAP and AMQP.

Figure 2.5 Shows the protocol stack for IoT systems

Figure 2.5 Protocol Stack for IoT Systems [67]

2.2.1.1 MQTT (Message Queuing Telemetry Transport Protocol)

MQTT is one of the popular M2M messaging protocols, which was developed by Andy Stanford-Clark of IBM and Arlen Nipper of Arcom Control Systems Ltd (Eurotech). It is based on public/subscribe mechanism and designed to support M2M communication for constrained devices [69]. MQTT client and MQTT broker are two important components of messaging system where, MQTT client publishes messages to an MQTT broker. MQTT broker acts as a subscriber for other clients. Each message is addressed to a topic[70]. Clients can subscribe to multiple topics and receives every message published to the each topic. It is binary protocol based on TCP transport protocol and uses TLS/SSL for security.

MQTT is a binary protocol and normally requires fixed header of 2-bytes with small message payloads up to maximum size of 256 MB [71]. MQTT protocol normally requires fixed header of 2-bytes with very low message payloads up to maximum size of 256 MB

[71]. MQTT is widely used for small constrained devices in a large networks where devices are controlled/monitored from back-end servers. MQTT is most suitable for large networks of small devices that need to be monitored or controlled from a back-end server on the Internet.

2.2.1.2 CoAP

CoAP is a Constrained Application Protocol which is a light weight M2M protocol proposed by IETF CoRE (Constrained RESTful Environments) Working Group. It supports both request/response and publish/subscribe architecture [69]. The architecture of CoAP is mainly designed to support both HTTP and RESTful web by using proxies. Unlike MQTT, CoAP uses URI (Universal Resource Identifier) instead of topics [71]. In CoAP, Publisher and Subscriber communicate through URI. Subscribers are notified, and new data as indicated by URI, when a publisher publish new data to URI. CoAP is also a binary protocol like MQTT and has a fixed header size of 4-bytes with small message payload depending on used technology or web service [71]. CoAP uses UDP, and DTLS for security [73]. Hence, clients and server communicate through connectionless datagrams with less reliability but higher data rate.

2.2.1.3 AMQP

AMQP refers to Advanced Message Queuing Protocol. This lightweight M2M protocol was developed by John O'Hara , JPMorgan Chase , UK in 2003. AMQP protocol is designed for interoperability, reliability, security and provisioning [68]. Like CoAP, AMQP also supports public/subscribe and request/response mechanism architectures [73].

It supports wide range of features for messaging like reliable queuing, topic based public/subscribe message mechanism, flexible routing and transaction [68]. Hence, AMQP provides connection-oriented communication between client and broker which makes reliable connection. AMQP offers two preliminary levels of Quality of Service (QoS) for messages delivery. These two QoS are Unsettle Format (not reliable) and Settle Format (reliable) [68].

2.2.1.4 HTTP

HTTP is a Hyper Text Transport Protocol which is popularly used a web message protocol. It was developed by Tim Berners-Lee originally. Later, further enhancement and development was done by jointly collaboration of IETF and W3C in 1997[7]. HTTP protocol works on request/response based RESTful web architecture. Similar to CoAP, HTTP also uses URI (Universal Resource Identifier) for message communication. Server and client communicate through URI. Client send request then server sends data through URI. HTTP does not define header size and payload as it is a text-based protocol instead it depends on the programming technology or webservices. HTTP uses TCP protocol and TLS/SSL for security [72]and provides connection oriented messaging between client and server. HTTP is a very popular and highly accepted web messaging standard globally as it offers several features such as persistent connections, chunked transfer encoding and request pipelining [72].Table 1 presents comparative analysis of four widely popular Messaging Protocols for IoT Systems: MQTT, CoAP, AMQP and HTTP

Table 2.2: Comparative Analysis of Messaging Protocols for IoT Systems: MQTT, CoAP, AMQP and HTTP [67]

Criteria	MQTT	CoAP	AMQP	HTTP
1. Year	1999	2010	2003	1997
2. Architecture	Client/Broker	Client/Server or Client/Broker	Client/Broker or Client/Server	Client/Server
3. Abstraction	Publish/Subscribe	Request/Response or Publish/Subscribe	Publish/Subscribe or Request/Response	Request/Response
4. Header Size	2 Byte	4 Byte	8 Byte	Undefined
5. Message Size	Small and Undefined (up to 256 MB maximum size)	Small and Undefined (normally small to fit in single IP datagram)	Negotiable and Undefined	Large and Undefined (depends on the web server or the programming technology)
6. Semantics/ Methods	Connect, Disconnect, Publish, Subscribe, Unsubscribe, Close	Get, Post, Put, Delete	Consume, Deliver, Publish, Get, Select, Ack, Delete, Nack, Recover, Reject, Open, Close	Get, Post, Head, Put, Patch, Options, Connect, Delete
7. Cache and Proxy Support	Partial	Yes	Yes	Yes
8. Quality of Service (QoS)/ Reliability	QoS 0 - At most once (Fire-and-Forget), QoS 1 - At least once, QoS 2 - Exactly once	Confirmable Message (similar to At most once) or Non-confirmable Message (similar to At least once)	Settle Format (similar to At most once) or Unsettle Format (similar to At least once)	Limited (via Transport Protocol - TCP)
9. Standards	OASIS, Eclipse Foundations	IETF, Eclipse Foundation	OASIS, ISO/IEC	IETF and W3C
10. Transport Protocol	TCP (MQTT-SN can use UDP)	UDP, SCTP	TCP, SCTP	TCP
11. Security	TLS/SSL	DTLS, IPSec	TLS/SSL, IPSec, SASL	TLS/SSL
12. Default Port	1883/ 8883 (TLS/SSL)	5683 (UDP Port)/ 5684 (DLTS)	5671 (TLS/SSL), 5672	80/ 443 (TLS/SSL)
13. Encoding Format	Binary	Binary	Binary	Text
14. Licensing Model	Open Source	Open Source	Open Source	Free
15. Organisational Support	IBM, Facebook, Eurotech, Cisco, Red Hat, Software AG, Tibco, ITSO, M2Mi, Amazon Web Services (AWS), InduSoft, Fiorano	Large Web Community Support, Cisco, Contiki, Erika, IoTivity	Microsoft , JP Morgan, Bank of America, Barclays, Goldman Sachs, Credit Suisse	Global Web Protocol Standard

The followings are the features which IoT should support to provide IoT services on the network.

1) **High connection density:** The objective of IoT is eventually to connect every possible object to internet. Thus, any object can be interconnected globally to the communication infrastructure.
2) **Multiple heterogeneous access networks:** IoT devices are based on different hardware platforms which can interact with other devices on different networks and hence heterogenous in nature. Thus, IoT must support heterogenous access networks.
3) **Autonomic networking:** IoT needs to support Autonomic networking which can include self-management, self-configuring, and self-optimizing etc. as per application areas.
4) **Security:** IoT is meant to connect every possible object hence, it raises a concern about security of data and services. A critical example of security requirements is the need to integrate different security policies and techniques related to the variety of devices and user networks in the IoT [5].
5) **Energy efficiency:** Sensor needs long life time activities, hence, energy efficiency plays an important role.
6) **Networking capabilities:** Providing efficient network control functions for network connectivity, authorization, mobility etc. is important in IoT.
7) **Transport capabilities:** IoT must support the connectivity for the transport of IoT service and application specific data information, as well as the transport of IoT-related control and management information [5].
8) **Local network topology management:** Support for management of local network topology by traffic management and network resource reservation for time-sensitive applications.

2.2.2 SDN Managed IoT Network Architectural Overview

Typically, an IoT network is comprised of devices like sensors, actuators, and smart devices. These are generally resource constrained devices and they try to connect to an access point or a Gateway as an access point to the Internet. These Gateways and access points provide connection to the IoT devices; the network from the access point to the Internet is termed as access network. Gateways are close to the user devices and are termed as edge devices. Traditional Gateways typically have less compute and storage capacities, but the modern IoT Gateways are being designed for more storage and compute to provide intelligence and analytics towards the edge. Supporting the access network is the Internet backbone network which routes the IoT data to data centers for further processing. Figure 2.6 presents an IoT network with reference to access, edge, backbone and data center networks. As we move towards the data center network, the storage and compute capacity increases but introduces latency [35].

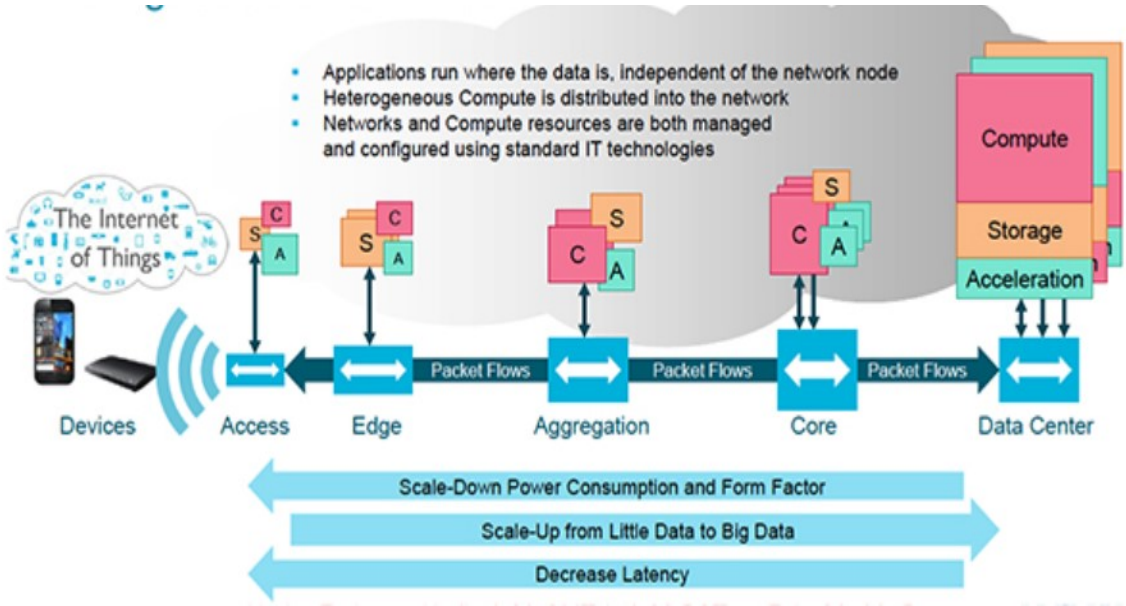

Figure 2.6 IoT network from edge to data center network [35]

A network where potentially a huge number of smart devices are inter-connected [2] using IoT technology will consequently have explosion in the amount of data streamed. Hence, efficient network management is the key for managing the huge amount of connected devices and the real-time data generated by them. Adopted as the best network management technology, SDN can play a vital role in managing IoT network at all layers, i.e., edge, access, backbone, and data center. IoT networks have many challenges and requirements to meet. Many requirements of the IoT applications can be fulfilled using SDN technologies, some of which are listed below:

1) **Network Management:** SDN is an efficient traffic engineering technology which could be useful to control the traffic flows in a dense IoT network. SDN technologies could be used for managing a IoT network by offering load balancing and efficient bandwidth utilization.
2) **Network Function Virtualization:** NFV makes devices capable of changing their functionality depending on applications in real time, which is favorable in IoT applications to manage devices to perform multiple tasks by changing network functions as per the application requirement [36] [37]. In large scale IoT networks SDN based technologies can support the realization of NFV [38].
3) **Application Wide Information Access:** The IoT device data must be accessible to the required applications and owners with authorization to hold a control over the devices and change device functions seamlessly. Such a control on devices over a network could be achieved with the help of SDN based technologies [39].

The above-mentioned points make it evident that SDN can play a major role in managing IoT networks and application services from the access network to the core data center networks. Considering the importance of SDN for managing IoT networks, the authors in [40] and [41] have proposed different SDN based architectures to support IoT network as per the use case.

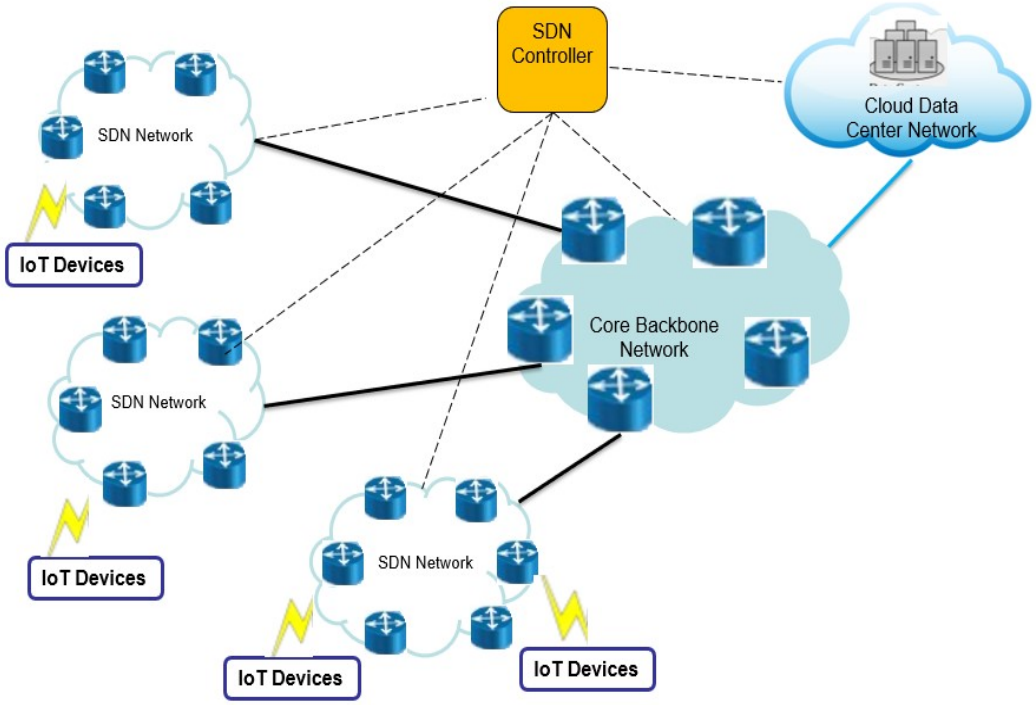

Figure 2.7 Architecture of SDN managed IoT

Figure 2.7 presents the SDN managed IoT network architecture with a centralized SDN controller. Each IoT network is connected to SDN switches which are managed by a remote SDN controller. The IoT data is routed by the SDN switches to the Cloud network through the core backbone internet. But such a centralized control design has several drawbacks [42], especially when the interconnected network becomes dense at the user

edge as in case of IoT networks. Many authors have proposed a distributed SDN controller architecture [43] [44] which overcomes the single point of failure scenario and provides better management control over the underlying network. In the distributed architecture, all SDN controllers are further managed and controlled by a remote master SDN controller. This is well suited to the IoT ecosystem, where the IoT network grows densely at the user edge and needs a more closer control management mechanism.

A distributed SDN controller architecture for the IoT network is shown in Figure 2.8, where SDN controllers are deployed closer to the edge network. A master SDN controller is deployed on the Cloud to manage distributed SDN controllers close to edge networks.

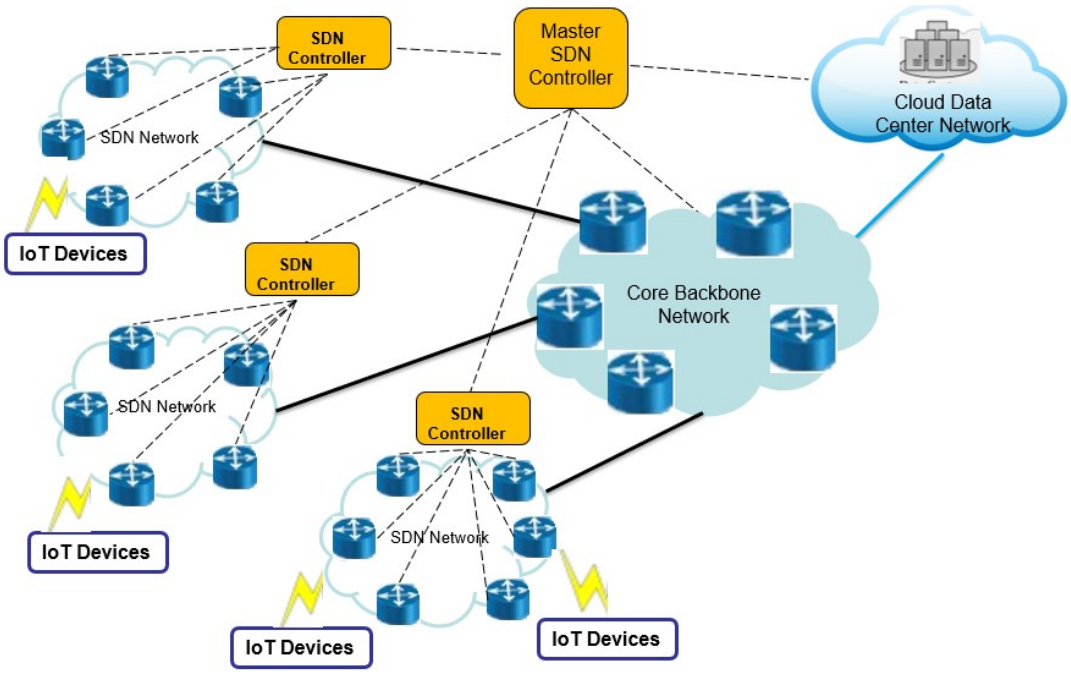

Figure 2.8 Architecture of distributed controllers in SDN managed IoT

2.3 Management Issues and Challenges in IoT

In any network, network management solutions are essential for managing devices and services. Traditional management functionalities, like remote control, monitoring, configuration and maintenance, plays a key role in IoT [19], but these traditional management functionalities are required to evolve to support heterogeneity of the IoT [20]. For example, management support to resource constrained devices such that it requires minimal energy for management operation is critical.

Research work [21] [22] describes the characteristics of IoT data to be categorized as: heterogeneity, inaccuracy of sensed data, scalability and semantics. Data management in IoT must also provide support for offline analysis [23]. Management systems needs to support heterogenous hardware devices operating on heterogenous networks locally and remotely. The managers can remotely control and troubleshoot the IoT devices which can reduce costs associated with maintenance tasks in less time. A remote-control management can accelerate the response time in failure situations. Thus, management system which supports the remote monitoring of sensors and actuators in remote location is very important in emergency applications [24].

The other management challenges include support for huge data volume aggregated by IoT devices and sensors and health monitoring. Management solution must also take care of the security of the data as well as the devices in the network by authorization and authentication. IoT devices are typically resource constrained hence, energy management is very important and challenging task. Table 2.3 describes the major management issues challenging the IoT.

Table 2.3- Management Issues in IoT [19]

Configuration Management	• How devices are set up • Network connectivity • Self-configuration capability • Asynchronous Transaction Support • Network reconfiguration
Monitoring	It is essential for the operation and control of devices to know their status, e.g., running, listening, down, sleep mode, etc. Therefore, once objects are deployed and in use, there should be a way to monitor their statuses. These are in addition to: • Network status monitoring • Network topology discovery. • Notification. • Logging
Maintenance:	Detecting the failure of a devices is important, specifically in an IoT network which might involve a larger number of devices. A tool or software is required for detecting and addressing device failure. Other issues relate to the general maintenance tasks of devices e.g. software update, patch update, protocols version detections, etc.
Control	Management issues including turning devices on and off, disconnecting from specific networks and connecting to other. To effectively control a device, a prior knowledge of their status is

	required. Therefore, "Device control" complements "monitoring of devices."
Performance	Monitoring the performance of devices is needed so sign of stress can be detected before the occurrence of any failure. This is significant for devices that might be deployed in remote locations, and essential in emergency applications [23], where availability and other QoS parameters are of high importance.
Security and privacy	There are basic security challenges such as authorization, authentication and access control that need to be addressed. Security bootstrapping mechanisms are also required. Other security issues are associated with device-to-device communications. For instance, if devices are to be accessed by applications or software independently from the human users, then there are security measures that need to be enforced to ensure that devices are not leaking information and disclosing private information to unauthorized applications or used miscellaneously. Thus, privacy is vital as well [24].
Energy Management	• Statistics on energy levels, e.g. estimated lifespan. • Management of energy resources.

2.4 Introduction to IoT Middleware and Related Work

IoT Middleware is a solution to some of the challenges stated in Table 2.3. Middleware acts as an interface between IoT end devices and applications. Middleware provides

connectivity between heterogenous IoT devices operating on heterogenous networks and the IoT applications and ensure effective communication.

The literature study and related work in IoT Middleware development categorize the existing architecture of IoT Middleware in the below class types:

1) **Service-based IoT Middleware:** These Middleware are based on the service-oriented architecture (SOA) and allows users to add or deploy a diverse range of IoT devices as services [25]. Figure 2.9 shows an architecture of a service-based IoT Middleware, which is composed of three-layer architecture adopted by the OpenIoT [26], an European Union project to standardize IoT platforms. This three-layer architecture is consisted of IoT devices as a physical plane, server resources and cloud services as virtualized service plane and IoT applications as an application plane [26]. Hydra and Global Sensor Networks (GSN) are examples of service-based IoT Middleware.

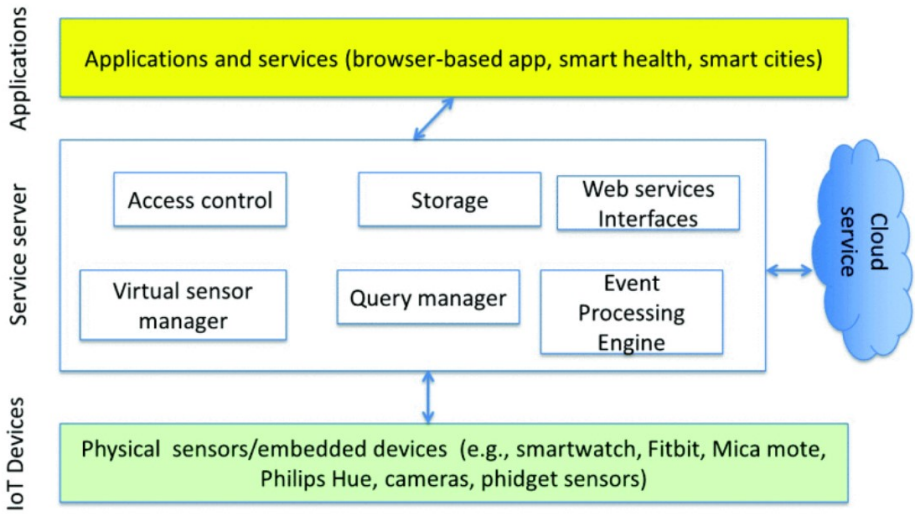

Figure 2.9 Service-based IoT Middleware [26]

In this type of architecture, a heavyweight Middleware is generally deployed on Cloud infrastructure or on powerful Gateways between IoT devices and the applications. The drawback of such the Middleware architecture is that the architecture is not designed to be deployed on resource-constrained IoT devices or edge gateways. It also doesn't support device to device communication in the network.

2) **Cloud-based IoT Middleware:** In this architecture the Middleware is deployed on cloud and the functionality of these Middleware is dependent on the chosen Cloud based platforms. This type of IoT Middleware solution restricts the use of IoT devices to be added to the network but allow more facility towards cloud application by allowing users to connect, collect, and interpret the collected data on cloud. The IoT devices and services can only be accessed through either application provided by vendor or Cloud supported RESTful APIs [26]. Google Fit and Xively are examples of cloud-based IoT Middleware. A Cloud based IoT Middleware architecture is shown in Figure 2.10.

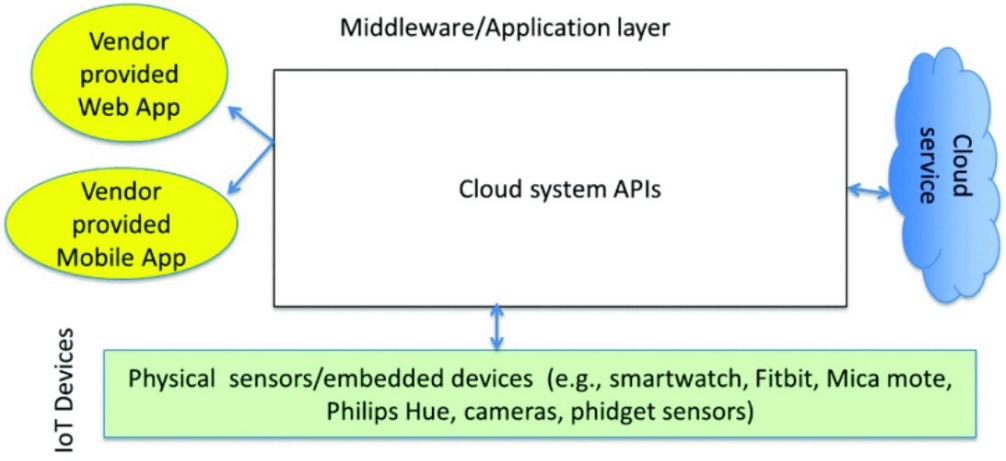

Figure 2.10 Cloud based IoT Middleware [26]

3) **Actor-based IoT Middleware:** This Middleware is based on plug and play architecture which supports a variety of IoT devices distributed in the network. TerraSwam [27], a joint research project between universities, government and private companies in USA, presented the actor-based IoT Middleware architecture [26]. The architecture is consisted of a layer of the sensory swarm of sensors and actuators [27]. The middle layer consists of mobile access such as smartphones, single board computers like Raspberry pi and laptops acting as gateway. Both the layers access the cloud services. This category of IoT middleware solution are lightweight and can be deployed on embedded devices, edge devices or cloud. A actor based IoT Middleware deployed on a sensor might not have storage service, however, an actor that provides a storage service can be downloaded from the Cloud repository when needed [27]. Calvin and Node-RED are examples of actor-based IoT Middleware. Figure 2.11 shows the architecture of the actor-based IoT Middleware [27].

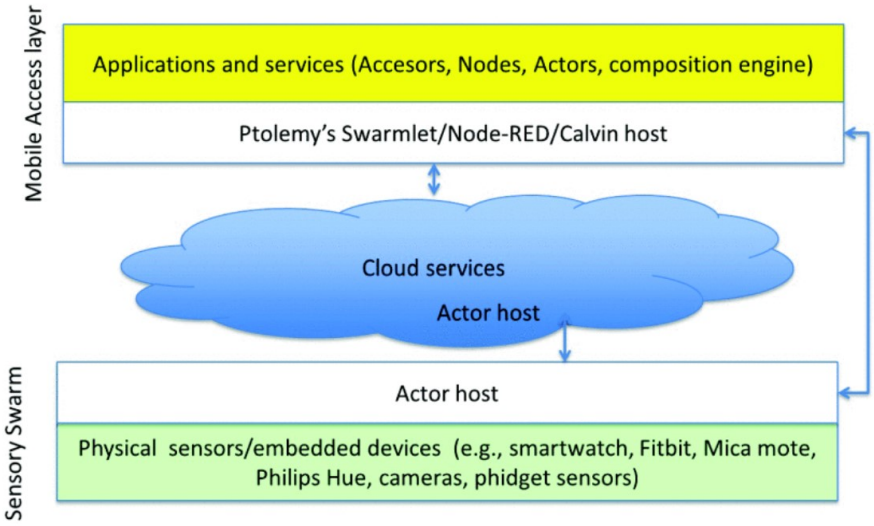

Figure 2.11 Actor-based IoT Middleware [27]

The main difference in all of the above three IoT Middleware architecture is in terms of supporting new diverse IoT devices, access to IoT devices and services and where

the Middleware can be deployed. The service-based IoT Middleware is designed to be deployed in the Cloud and provides simple web application to view raw IoT data. It provides very minimal support for integration with multiple applications. This architecture doesn't support the computational units to be changed by the users as per preference. While the actor-based IoT Middleware can be deployed in sensor layer, middle layer or cloud application layer as per IoT application area , thus provides scalability and reduced latency. It enables extension of computational units [27]. In the Cloud-based IoT Middleware architecture users need to trust the Cloud provider for data security and integrity and can store their data only as provided service by the provider. Even the Middleware service can stop functioning in cloud-based Middleware if the cloud service provider ends the service and the data could be lost. An example of the Cloud-based IoT Middleware is Google Nest [28]. On the other hand, in service-based Middleware and actor-based Middleware, users can choose where they want to store the data and how they want to store it.

The existing IoT Middleware systems are typically service based which supports efficient and secure processing [26] of streaming data from similar IoT devices, while Middleware which are concerned on providing usability to users are typically Cloud-based IoT Middleware [27].

2.5 Key Challenges and Issues in designing IoT Middleware solution

IoT Middleware can help in realization of IoT and development of diverse IoT applications and services however, it also poses a number of challenges [29], [30], some of which are discussed as follows:

1) **Middleware Development and Deployment:** The main challenge for IoT Middleware is the development of the Middleware in such a way that it can be deployed both in Cloud as well as Edge devices like Gateways and IoT devices to support diverse applications related to deep analytics or latency sensitive. The middleware needs to be lightweight and portable to be deployed at any layer of IoT network.
2) **Context based IoT Applications:** Another main challenge is to enable users to create their own IoT applications as per the context. The Cloud based IoT Middleware restricts the users to use the preprogrammed IoT applications provided by the service provider hence the users cannot create their own innovative applications. While the service-based architecture provides user with a SDK tool with low level programming and limited functionalities hence restricts development of large IoT applications as per user needs [26].
3) **Service Discovery**: Another major challenge is the service discovery in the evolving IoT network. In IoT network many devices and services could go live anytime while some could get unavailable, hence it's important to be able to query and discover devices and required services. Critical applications like health monitoring should not be get impacted while replacing any failed device or service in the IoT network.

2.6 Comparison with Related Work

Many research efforts and surveys have been conducted on the Middleware technology for IoT. Majority of existing research work in IoT Middleware is based on wireless sensor network. Few research papers are concentrated on cloud-based and service-based IoT

Middleware architecture while a very small number of research work in IoT Middleware solution is focused on IoT hardware resources or multiple protocol support.

Research paper [31], proposes the main functional components for service-based IoT Middleware which includes interoperability, IoT device discovery and management, semantic service discovery, context-based detection, security and privacy, and persistent storage management. Based on these functional components, Bandyopadhyay et al. [31] provided a survey of classification of existing IoT Middleware systems which concluded that these Middleware systems showed least support for functional components including interoperability, context-awareness, security and privacy. Interoperability between heterogenous IoT devices, heterogenous network and protocols and diverse applications is very difficult to achieve and is a major challenge in the development of IoT Middleware solutions.

Research work by Chaqfeh and Mohamed [32] presents concluded list of the major challenges including interoperability, scalability, abstraction, infrastructure support, security and privacy by investigating IoT Middleware solutions. The authors also concluded that semantic Web service based IoT Middleware solution is capable to address these challenges except scalability of persistence service [26]. It is a feasible solution to support device and service discovery however the web service based IoT Middleware are tend to be heavyweight and are not an ideal solution for deploying Middleware on resource-constrained Edge devices in IoT. For evolving IoT network it has become very important to develop a Middleware solution which is lightweight and can be deployed both in Cloud or in Edge devices as per the application. A latency sensitive application like autonomous driving may need Middleware to be deployed on sensor nodes or edge devices while

applications which require higher computing and analytics would favor a Cloud based Middleware deployment. Research work presented by Teixeria *et al.* [33] is also focused on the service-based IoT Middleware and proposes a solution for device and service discovery. The solution is based on probabilistic approach to deal with scalability, service discovery, device discovery and semantic data.

A more recent survey [34] provides an overview of IoT Middleware solutions and their requirements with challenges. Author Razzaque *et al.* grouped the requirements into functional and nonfunctional requirements and architectural requirements. However, the research work lacks to provide enough detail about each challenge and how they have been addressed by the existing IoT Middleware solutions.

Research work [26] summarizes the most popular existing IoT Middleware and their functionalities, as shown in Table 2.4. Each column of the table presents current challenges faced by the IoT Middleware solutions and each row mention the functionalities provided by the existing IoT Middleware system for the IoT applications. The functionalities of the existing IoT Middleware is compared to analyze the strength and weakness of the current IoT Middleware solutions and hence which areas need to be focused for future advancements.

Existing IoT Middleware systems including Hydra, GSN, Google Fit, Xively, Paraimpu, Calvin, Node-RED, Ptolemy Accessor Host are compared against Middleware functionalities [26].

Table 2.4 IoT Middleware Versus Functionalities [26]

Middleware	Functionalities							
	Device abstraction	Network connectivity	Composition	Monitoring and visualization	Service discovery	Security and privacy	Persistency	Stream processing
Hydra	Web service	ZigBee, Bluetooth, RFID, Wi-Fi, HTTP, SOAP	SDK took kit with Semantic Model Driven Architecture	Vendor GUI application	OWL and SAWSDL for device description	Distributed security and social trust	Unknown	Event manager
GSN	Virtual sensor/XML deployment descriptor	Push/pull, SOAP, HTTP	Specific application created by programmer	Vendor Web application	Key-value store registry for devices	Login account, electronic signature	SQL database	SQL query on data stream
Google Fit	Fit Programmable API or RESTful resource	Bluetooth, Wi-Fi. HTTP	Fitness based devices and data	Vendor Web application	Not supported	Permission and user control module	Cloud storage	Sensors API
Xively	RESTful resource	Bluetooth, Wi-Fi, Socket, MQTT, HTTP	Not supported	Vendor Web application	Not supported	User control on how data is shared	Cloud storage	RDF stream processing
Paraimpu	Sensor, Actuator, Connection	Pull/push, HTTP	Compose one sensor with one actuator using Connection	Paraimpu workspace	New sensor can announce its presence	Access token needed for access, user data shared according to the stated privacy law	Mongo DB	JSON query
Calvin	Actor	Wi-Fi, BlueTooth, i2c	CalvinScript	Vendor Web application	Provide search via Calvin Control APIs	Not supported	Distributed Hash Table (DHT)	Data flow processing
Node-RED	Node	MQTT	Drag and drop visual tool	Browser-based flow editing tool	Key word based search provided for node	User permission with password securely hashed using the bcrypt algorithm	Local file system	Node streams
Ptolemy Accessor Host	Accessor	MQTT, Bluetooth, Wi-Fi. HTTP, UDP, CoAP, Websockets	Drag and drop visual tool based on Virgil	Java-based GUI application	Import and export facility	Key management and Taint analysis	Local file system	Discrete event director

In the first column of Table 2.4, device abstraction refers to concepts and technology used encapsulate the heterogenous physical devices. The next column is network connectivity, which summarizes the supported communication protocols, example ZigBee, Wi-Fi, HTTP etc. The next column is the composition, which refers to the ability to connect to different vendors IoT devices. The monitoring and visualization column summarizes the user interface capability to monitor state of the devices. The next column is the service discovery, which shows the ability of Middleware to discover services. Next is the security and privacy column, which shows measures taken to securely store and retrieve data while the persistency column details the persistent storage used. The last stream processing column compares the middleware based on the data stream processed by the Middleware.

In conclusion, none of these existing research work and surveys addresses the more recent trend of light-weight plug-and-play Fog-based IoT Middleware. This thesis research attempts to explore the potential benefits of light-weight IoT Middleware which can be deployed on Fog in distributed architecture to empower users to create innovative IoT applications targeted for ambient data collection and analysis. This thesis attempts to explore the Middleware solution best suited for both low latency applications (Edge) and deep analytics (Cloud) in a SDN managed IoT network. The detailed design of the SDN managed IoT network testbed is presented in Chapter 3 and the results of the experiments is discussed in Chapter 4.

2.7 Chapter Summary

Network Softwarization is a concept that refers to the Softwarization of network resources to enhance efficiency by software programmability and reduce dependability on hardware. Network Softwarization focuses on key technologies like SDN, NFV, IoT and Cloud. Network Softwarization technologies are changing the current network design and architecture by employing software components and IoT application areas can benefit from these technologies.

There are many management and technical challenges to the realization of IoT systems and a solution to these challenges is Middleware. Middleware is a software system between devices and applications which can provide interoperability and abstraction. There has been many researches and surveys conducted to propose the design and implementation of Middleware based on application, agent, services based architectures. Some have advantages over other solution, but specific to a use case or application. There is no literature available on Middleware solution which can used as a multifunctional system, firstly to be efficiently deployed both on resource constrained Edge device like IoT Gateway, as well as on Cloud to support both local and deep analytics. Secondly, which can also solve the issue of interoperability between different heterogenous IoT devices operating on heterogenous communication networks and different application services build over different IoT data protocols. And thirdly, operate in SDN managed IoT network to take benefits of SDN technologies. Taking motivation from these challenges a Middleware solution is proposed in Chapter 3.

Chapter 3

Proposed Design and Implementation of IoT Middleware

This chapter describes about the proposed IoT Middleware which could be deployed both on Edge and Cloud network. Section 3.1 discusses current IoT needs and concerns to be addressed by IoT Middleware and Section 3.2 describes how a Middleware solution can be used in SDN managed IoT networks. A comprehensive description on the proposed IoT Middleware model design is presented in Section 3.3. The designed Middleware acts as a broker to the subscribed IoT applications. Utilizing the benefits of SDN and the Middleware solution a proposed distributed IoT Middleware system architecture is described in Section 3.4. Section 3.5 presents the experimental testbed setup designed for the proposed architecture along with the implementation tools. Next, Section 3.6 presents the device discovery module in our proposed architecture. This module works in two phases, firstly Network Scanning for heterogenous networks on which IoT sensors and devices communicate with the IoT Gateway. The second phase is a new node detection and node removal detection process for Topology modelling.

3.1 Need for IoT Middleware

The objective of the Middleware development is to develop a framework which can enable an adaptation layer in a plug-n-play mode. IoT Middleware is the software that must exist between physical devices (sensors, actuators, relays, etc.) or data endpoints, and the higher-

level software applications like artificial intelligence, predictive analytics, and cognitive computing. IoT Middleware moves data between the physical and digital realms and provides software resources to cope with the data generated from the IoT devices.

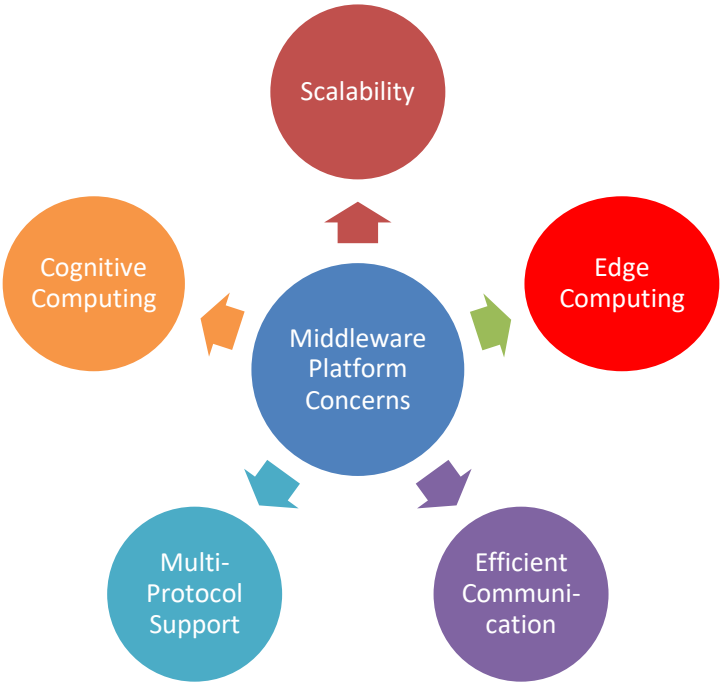

Figure 3.1 Concerns for Middleware platform design

While Middleware is not a new technology, but the current rise in IoT application areas has demanded a need for a new type of Middleware platform that addresses new concerns, as shown in Figure 3.1, including:

1) **Scalability:** The massive volume of IoT devices and other data endpoints involved in IoT applications requires advanced and flexible scalability. IoT Middleware needs to be efficient to support data from a large number of devices, all trying to exchange not

only information with a central hub server or an application, but also with each of the other devices in the application. Addressing, configuring, and managing thousands—even tens of thousands—of devices must be accounted for.

2) **Edge Computing:** Centralized intelligence and control topologies are being reevaluated in favor of the distributed architecture, with intelligence pushed into each edge device or data endpoint. IoT applications involve intercommunication between thousands of IoT devices for continuous monitoring, communication and control between devices. Edge computing uses intelligence at the edge of the network to decrease network latency, deliver real-time control and monitoring, and offer report by exception to reduce data volume.
3) **Efficient IoT Communication:** Usually IoT applications face challenges with unreliable networks and low-bandwidth. Hence, a requirement for an efficient network and communication architecture to support IoT network is emerging. Data communication protocols such as Advanced Message Queuing Protocol [AMQP] [45], Constrained Application Protocol [COAP] [46] and Message Queuing Telemetry Transport Protocol [MQTT] [47] are lightweight protocols and are based on the widely used publish/subscribe architecture. These protocols can reduce network latency and improve real-time communication speed between devices and endpoints.
4) **Multi-Protocol Support:** A Middleware need to support not only data protocols like HTTP, AMQP, COAP, MQTT etc. but also communication protocols like Bluetooth, Wi-Fi, ZigBee, cellular, RFID, 6LoWPAN etc. and other protocols including discovery protocols, security protocols and management protocols.

5) **Cognitive Computing:** The key value-added of IoT applications is predictive analytics. Knowing when a part is going to fail before it actually fails can bring almost immeasurable value to some IoT applications through reduced truck rolls, safety improvements, and improved overall equipment efficiency. The basis of predictive analytics is cognitive computing, essentially computers that mimic the way the human brain works. For IoT Middleware to perform predictive analytics, it will need support for cognitive computing.

Figure 3.2 shows the functional components of an IoT Middleware [48] in terms of IoT devices, interface protocols, abstraction and application service.

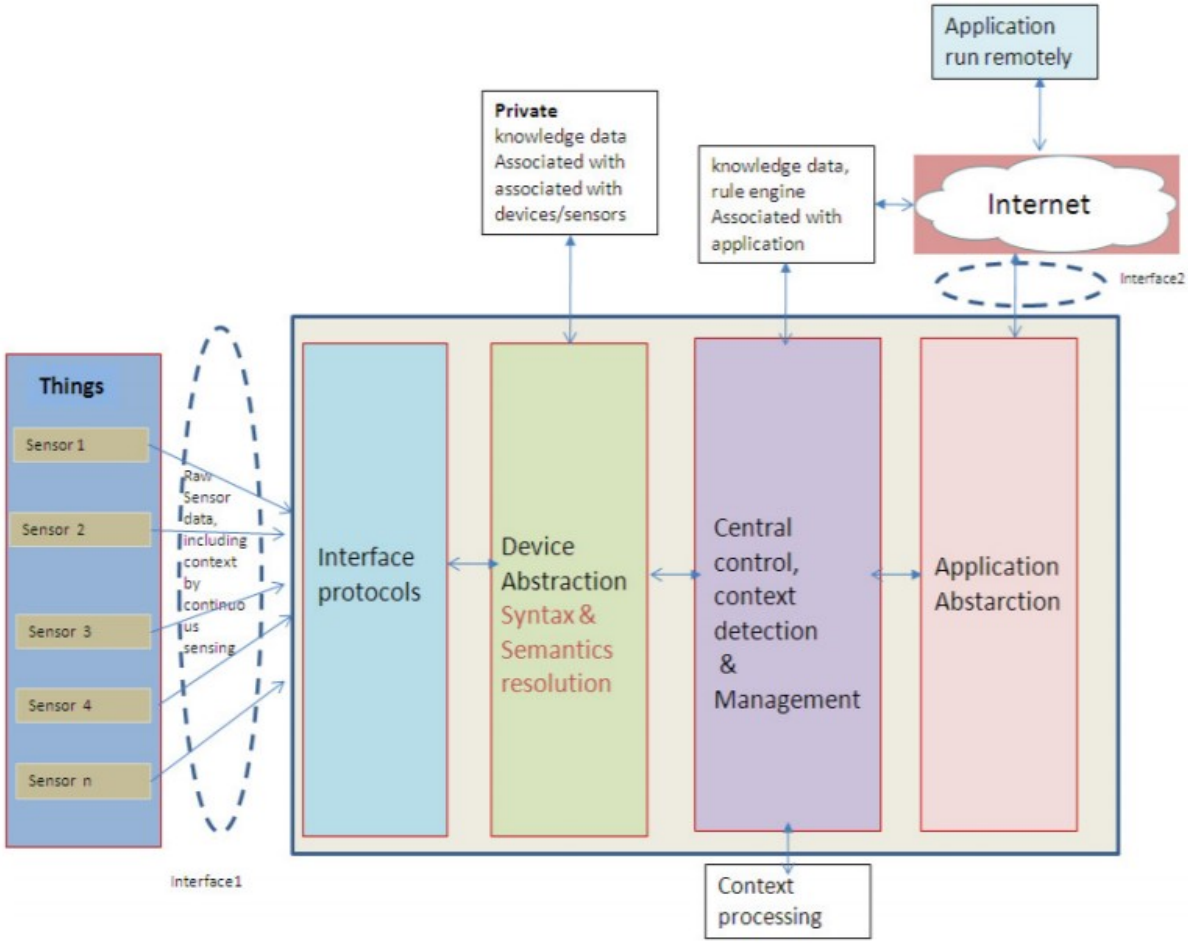

Figure 3.2 Functional components of IoT Middleware [48]

3.2 IoT Middleware for SDN managed IoT Network

OpenDaylight (ODL) [49] is a collaborative open source project hosted by the Linux Foundation with a goal to promote SDN and NFV. ODL is a modular open platform for controlling and automating networks. In the recent advancement the ODL community has designed IoT for Data Management (IoTDM), a Middleware solution for the IoT Network leveraging SDN capabilities [50]. IoTDM acts like a IoT broker for accessing information. It has protocol binding for HTTP, COAP and MQTT and can include more IoT data communication protocols. The advantage of using this model is the availability of plugin connectivity to the ODL SDN controller for using SDN technology in IoT Network. The model has been primarily designed as a Cloud Middleware. As a result, IoTDM Middleware lacks some features necessary to be hosted near edge. One of the limitations is that IoTDM Middleware only supports IP Protocol binding which makes it more suitable for backend infrastructure. Figure 3.3 shows the block diagram of OpenDaylight IoTDM Middleware [50].

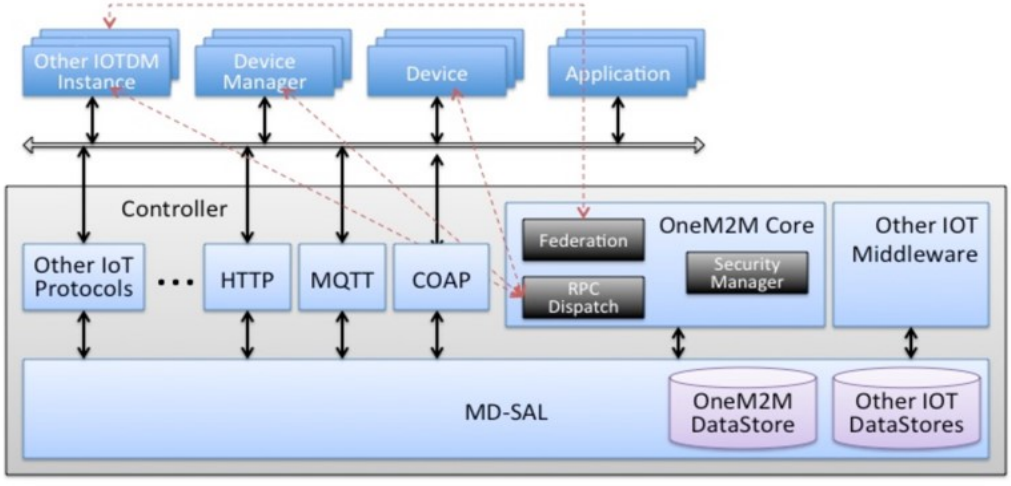

Figure 3.3 Block diagram of OpenDaylight IoTDM [50]

3.3 Proposed IoT Middleware Design

This thesis explores the potential of the distributed architecture of the Middleware solution in SDN managed IoT networks. The goal is to design a Middleware which can be deployed close to the user edge with enhanced capabilities like analytics and intelligence. The Cloud Middleware is designed as a backend solution with binding to communication protocols working on the IP technology. But to deploy a Middleware closer to the user edge, the Middleware must support multiple protocol bindings and it must be able to do multi-protocol translations to provide a horizontal unified data integration scheme for interoperability for applications on the Cloud and IoT devices at the edge. With this concept a light weight Middleware model is designed in this thesis, which has intelligence and multi-protocol bindings to be deployed on IoT gateways where diverse devices and multiple protocols coexist.

Figure 3.4 shows a block diagram of the proposed Middleware which can be deployed on resource constrained IoT Gateways. The Middleware solution has multiple protocol binding on unified protocol interface as the southbound interface for diverse IoT devices using Wi-Fi, Bluetooth, serial, or other protocol. The data from each device is stored in the data store in JSON format. A SQLite database is used to store the sensor data. SQLite is a serverless database, hence support the objective to achieve lightweight Middleware solution on resource constrained devices like Raspberry Pi 3. However the proposed Middleware can also be deployed on a more powerful device than Raspberry Pi 3 (as IoT Gateway) for IoT applications which are memory intensive.

A unified API interface provides access to IoT data using API calls to the broker. Middleware has bindings for HTTP, COAP, MQTT and AMQP protocols. HTTP and

COAP are based on the request and response model, and require post methods to publish IoT data to the respective applications. The MQTT and AMQP are based on publish /subscribe model where MQTT/AMQP based IoT devices publishes data to the communication broker under a topic. The MQTT/AMQP applications subscribed to the topic receives the relevant data. RabbitMQ AMQP broker is installed to support AMQP messaging between publisher (IoT devices) and applications running on the Cloud or other IoT devices. A MQTT plugin is installed with RabbitMQ to support intercommunication between the MQTT and AMQP publisher and subscribers for interoperability.

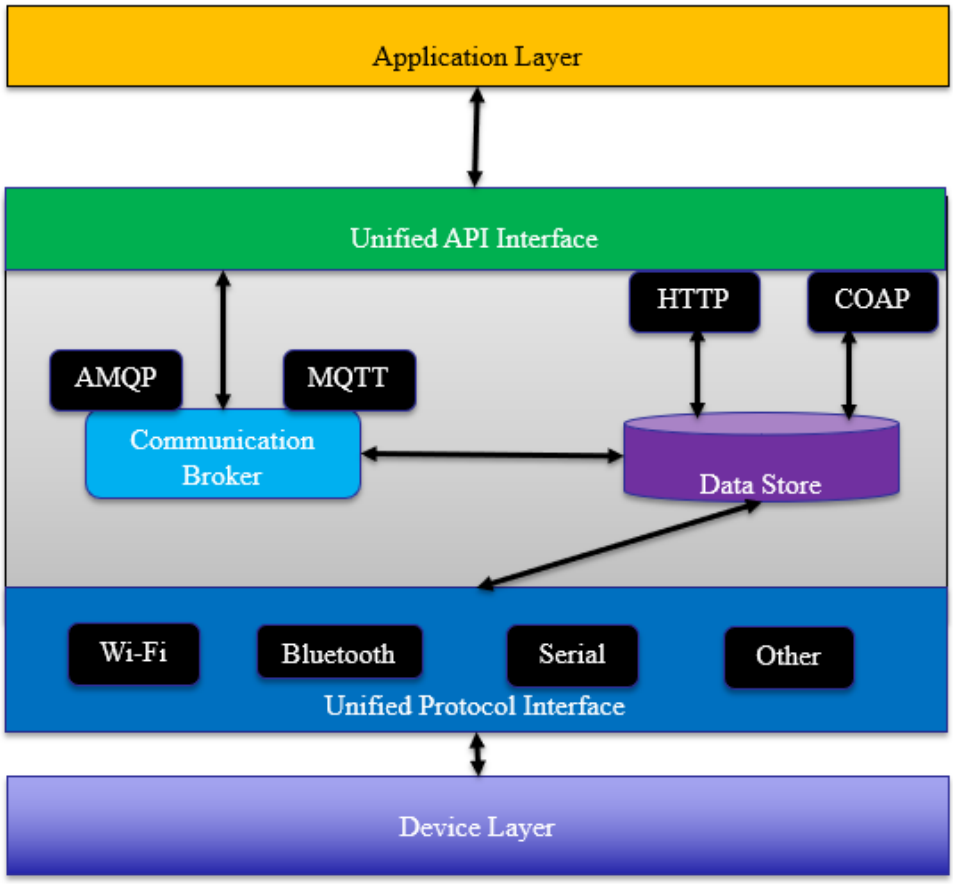

Figure 3.4 A block diagram of proposed Middleware solution

When a AMQP publisher (IoT device) transmits data to the RabbitMQ broker it uses a queuing mechanism to match and send data to the authorized subscriber application. A copy of the received sensor data is also stored in the middleware repository which can be accessed by REST-based services like HTTP and COAP. Since the broker is installed with MQTT plugin, it supports interoperation between AMQP and MQTT. Thus, in the proposed model, as shown in Figure 3.4, an AMQP based device data item can be easily accessed with an HTTP or COAP, or AMQP, MQTT application, thus providing interoperability between devices and applications by abstraction and translating underlying communication and data protocols.

The proposed Middleware is developed using the Python language and uses different libraries and packages for developing different protocol bindings and Python algorithm for data processing. The following highlights the design components:

1) **Uniform Communication Protocol Interface**

   An algorithm is implemented using Python with the help of PyBluez 0.22 Python library for sensors that communicate using Bluetooth. Similarly, other algorithms are developed in Python for serial communications and Wi-Fi communications. We have used python-Wi-Fi 0.6.1 library module for developing the algorithm for those sensors using Wi-Fi and pyserial library for developing the algorithm for the serial communication interface.

2) **Uniform API Interface**

   Unified API Interface supports different modes of communication like COAP, HTTP, MQTT and AMQP. We have installed amqp 1.4.9 Python library [51] and developed an algorithm in Python using this client for communications with the AMQP protocol.

Similarly, for communications using the MQTT protocol, we have installed Eclipse Paho MQTT Python client library, paho-mqtt 1.3.1 (latest version support python 2.7 and 3.x both) [52]. We developed the python algorithm by using paho-mqtt client library for communication using MQTT protocol. We have also installed python client library openwsn-coap 0.0.2 [53] for communications using the COAP protocol. For communications using the HTTP protocol, we have installed urllib3 python library. urllib3 [54] is a powerful HTTP client for Python. Much of the Python ecosystem already uses urllib3. urllib3 brings many critical features that are missing from the Python standard libraries like thread safety, connection pooling, client-side SSL/TLS verification, file uploads with multipart encoding, helpers for retrying requests and dealing with HTTP redirects etc.

3.4 Proposed System Architecture for Distributed IoT Middleware in SDN managed IoT network

Taking reference from the aforementioned technologies and solutions, a SDN-based IoT network testbed has been designed for conducting experiments and evaluating the proposed Middleware distributed architecture. A network slice of the complete network architecture shown in Figure 2.8 is used to design and implement the SDN based IoT network testbed for the thesis. Figure 3.5 shows the network slice highlighted and Figure 3.6 shows the SDN managed IoT network architecture obtained from the slice where SDN controller is a local controller controlled by the master SDN controller.

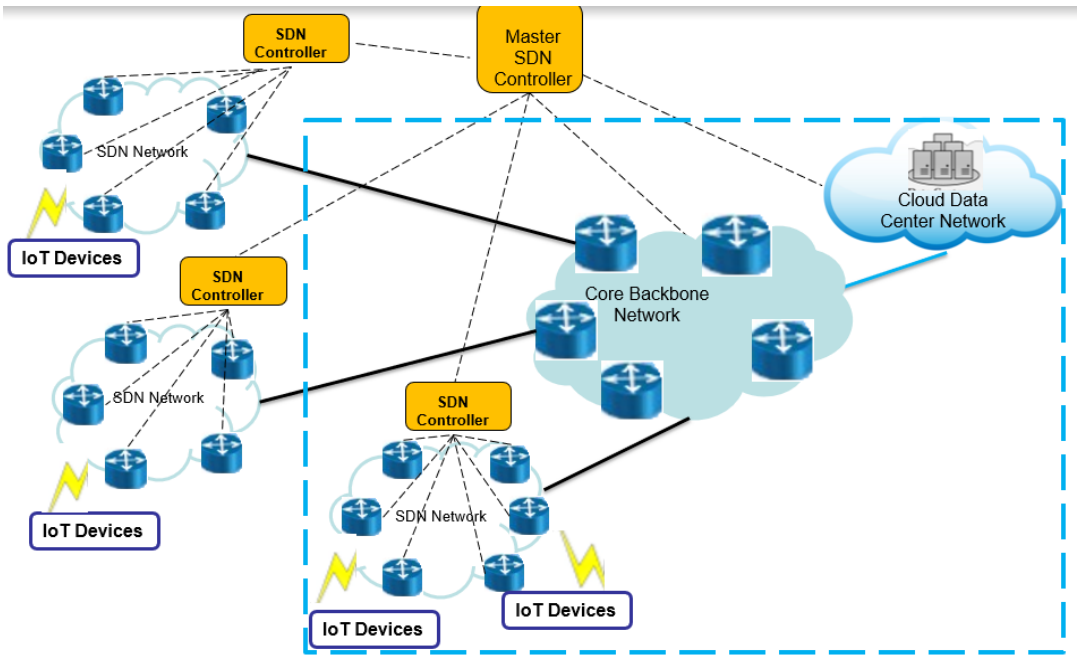

Figure 3.5 Highlighted network slice of the distributed SDN managed IoT network architecture is used for designing testbed

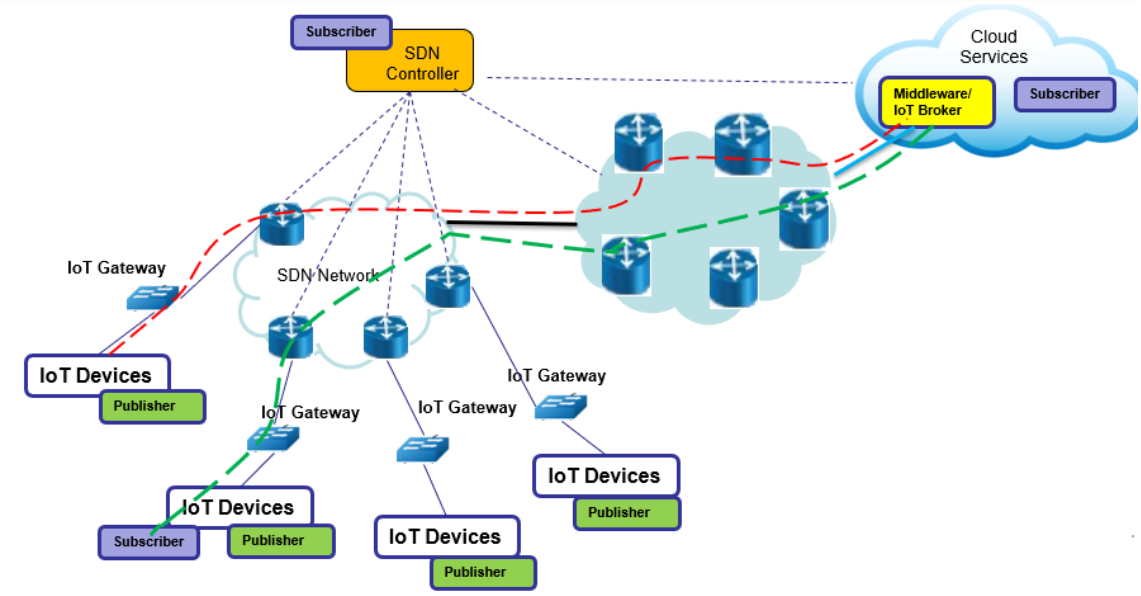

Figure 3.6 Architecture of SDN managed IoT network where Middleware is deployed on the Cloud

Figure 3.6 shows the architecture of SDN managed IoT networks where the IoT Middleware is deployed on the Cloud. This architecture is latency prone to the IoT applications requiring near real time analysis. An example is an autonomous vehicle application where latency in critical situation can lead to accidents. Such applications require data to be processed in close proximity to the user edge for better intelligence and decision making. Thus, in this thesis an extension of this architecture is proposed where the Middleware is also deployed on each IoT Gateway for a closer proximity to the user applications. The developed experiment testbed is based on the modified architecture, as presented in Figure 3.7. In this architecture, the IoT device subscriber can access data from a nearby IoT Gateway Middleware / broker instead of the remote Cloud Middleware / broker for real time data streaming.

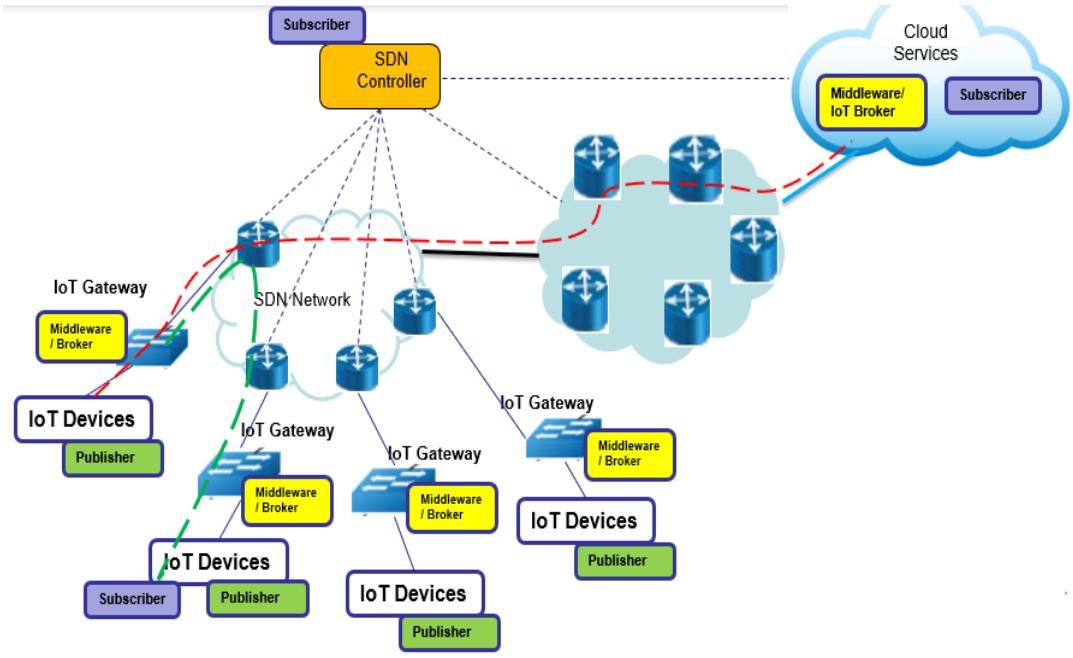

Figure 3.7 Architecture of software-defined IoT where Middleware is deployed on IoT Gateways and the Cloud

Figure 3.7 focuses more on control towards the edge network. The components used in the design are explained as follows:

1) **Sensors:** Sensors usually have very limited network connectivity capability. The majority of sensors in the market utilize Bluetooth Low Energy (BLE) and some sensors use the Zigbee Protocol along with some bunch of other protocols for LAN and PAN. But all of these have a limitation, i.e., they can not directly connect to the WAN or the Internet. To make these sensors or sensor network connect to the Internet, the Gateways play an important role.
2) **Gateway:** A Gateway provides a single point of contact for multiple services. Gateways can be designed to connect to sensors working on different protocols and initiate protocol translation to connect to the Internet using Wi-Fi, Ethernet or Cellular communications. Gateways are placed at the intersection of sensors, devices and the Cloud. Traditional Gateways were designed mostly to perform protocol translation and device management functionalities. They were not built for complex processing, analytics and intelligence in depth. Figure 3.8 shows the traditional Gateway.

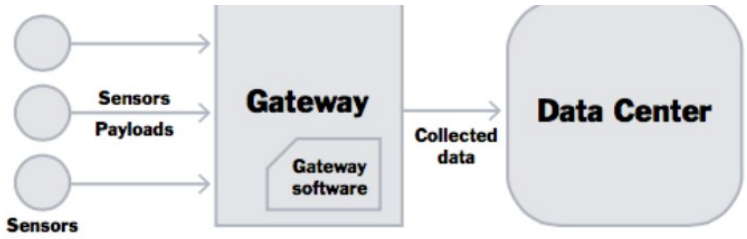

Figure 3.8 Architecture diagram of traditional Gateways [55]

The current generation Gateways for IoT are designed to support intelligence at the Edge network. They can perform several functions including

connecting to heterogenous devices, translate protocol, filter, aggregate and process IoT data, provide management support for updates and security, etc. They can also serve as a platform for applications to processes data and provide intelligence at the Edge network supporting IoT. Figure 3.9. shows the architecture diagram of the current generation IoT Gateways.

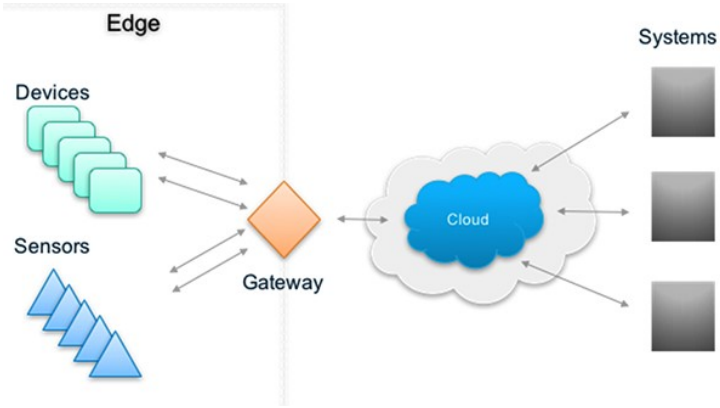

Figure 3.9 Architecture diagram of today's IoT Gateways

3) **Middleware:** The proposed Middleware solution is deployed on the IoT Gateway to bring processed data and intelligence closer to the user edge. This approach is suitable for near real-time applications.

4) **Switch:** The designed IoT network is managed by SDN switches for forwarding IoT data traffic as per flow rules. The Gateway devices extend connection to SDN switches to connect to the Internet applications.

5) **SDN Controller:** Multiple SDN controllers in distributed architecture supports the multiple IoT networks. Each SDN controller has been placed close to the edge network for more control and programmability of business logic close to edge. This approach makes the edge network smarter for data processing, analytics and intelligent decisions.

OpenDaylight SDN controller has been used in the design to control traffic through SDN switches.

6) **Cloud:** For the Cloud network, the Microsoft Azure Cloud services are used. Dashboard [56] applications are developed on the Azure Cloud to stream IoT data from sensors to the Cloud. The proposed Middleware solution is also deployed on the Azure Cloud to test the performance analysis of the proposed Middleware on Cloud and on IoT Gateway close to user edge.

3.5 Experimental Testbed Setup

Figure 3.10 shows the experimental testbed developed to explore the potential of the proposed Middleware solution on IoT Gateways and on the Cloud. A detailed description on each implementation tool is presented in the following sub-sections.

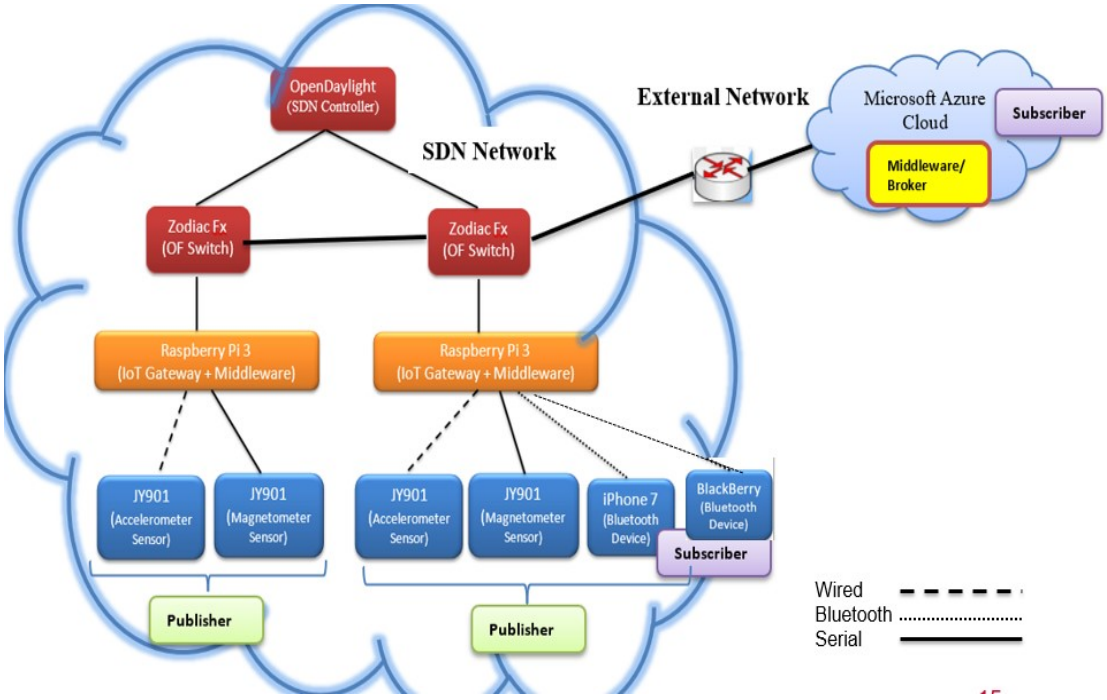

Figure 3.10 A schematic view of SDN managed IoT network

3.5.1 Implementation Tools

A significant time has been devoted in the initial investigation of the resources required to build a testbed for SDN enabled IoT network. Table 3.1 and Figure 3.11 provides the detail of hardware resources used in the testbed.

Table 3.1 Detail of hardware resources used to build SDN enabled IoT Testbed

Device	Hardware	Units
Sensors	JY901	4
IoT Gateway/ Middleware	Raspberry Pi 3	2
SDN Switch	Zodiac Fx SDN switch	2
SDN Controller	ODL (Ubuntu Machine)	1
Bluetooth Device	Iphone 7, BlackBerry Priv	2

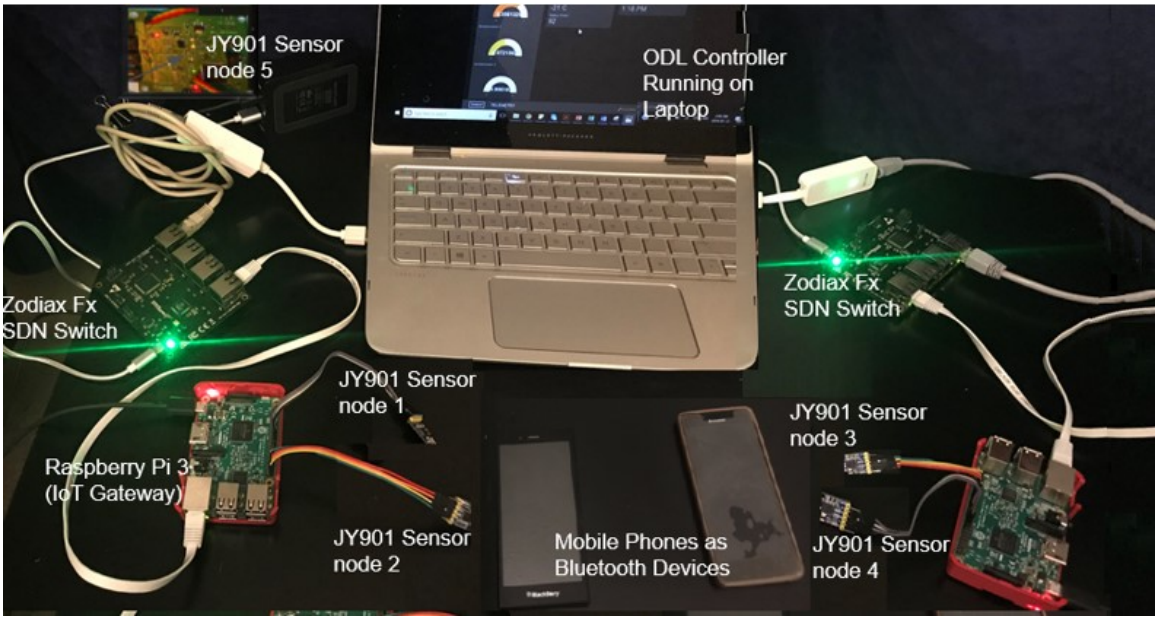

Figure 3.11 Experimental Testbed for SDN managed IoT network

A description for each hardware resource used in testbed is presented as follows:

1) **JY901 as Sensors:** We have chosen to use JY901 sensor node which has accelerometer, magnetometer and gyroscope with extended functionality using GPS. This type of sensor is suitable to track the movements of the objects. JY901 sensor specification [57] are as below:
   a) Gyroscope in JY901 measures angle, acceleration and angular velocity
   b) It has a 3-axis gyro and a 3axis accelerometer and a 3axis magnetic.
   c) It uses Kalman filter combining with the gyro and accelerometer to get high precision of the angle measurement
   d) It can be used such as four-axis flight control and self-balancing robot and the module can get accurate attitude in a dynamic environment with precision is 0.05degrees.
   e) It uses high-precision gyro accelerometer MPU6050, which read the measurement data by serial port and IIC port. The sensor uses advanced digital filtering technology with minimum noise and improves accuracy.
2) **Raspberry Pi 3 as IoT Gateway node:** A Gateway node must be able to connect to different sensors and actuators using multiple communication protocols both wired and wireless. Further as the Gateway must be able to translate protocol to connect the sensor network to the Internet and the Cloud. As described in Section 3.3, today's IoT Gateways need to be smart for data processing and applying business logic and intelligence in order to support edge analytics. To meet these high demands, an IoT Gateway must have good storage, processing power and battery power support. These criteria are well fulfilled by the new generation compact Raspberry Pi 3, as depicted in

Figure 3.12, which is a credit card sized single board powerful computer for IoT solution. Raspberry Pi 3 has 4 × ARM Cortex-A53, 1.2GHz CPU with 1GB RAM and microSD supported expandable storage up to 32 GB. It has support for 10/100 Ethernet, 2.4GHz 802.11n wireless networking with inbuilt Bluetooth 4.1 Classic, Bluetooth Low Energy. It has multiple ports including HDMI, 3.5mm analogue audio-video jack, 4× USB 2.0, Ethernet, Camera Serial Interface (CSI), Display Serial Interface (DSI).

Figure 3.12 New generation Raspberry Pi 3 [58]

3) **SDN Switch:** To design the SDN network, Zodiac Fx Switch [59] has been used which is based on OpenFlow. Zodiac Fx is the first OpenFlow switch developed by Northbound Networks to build and test real time SDN applications for research and labs. Figure 3.13 shows the Zodiac Fx SDN Switch.

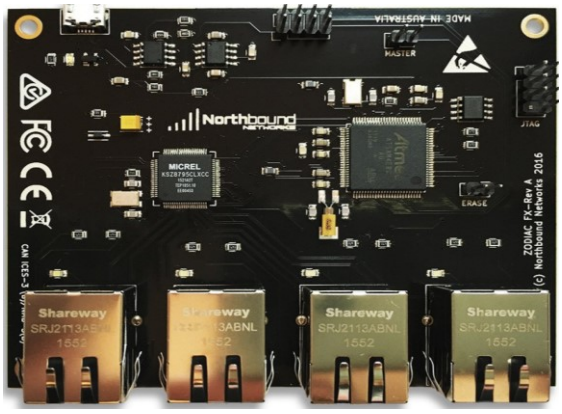

Figure 3.13 Zodiac Fx SDN Switch [59]

Zodiac FX is controlled by an Atmel CPU. It has 4 ports and works with the OpenFlow protocol Version 1.3, and it comes with support for open source firmware. Features of Zodiac Fx are presented as below:

a) SDN / OpenFlow controllers compatibility: OpenDaylight, Floodlight, RYU, OpenMUL, HP VAN Controller
b) Microcontroller Atmel ATSAM4E8C.
c) 4 10/100 Fast Ethernet ports
d) Support for OpenFlow 1.0, 1.3 & 1.4
e) 512 entry software flow table
f) 64KB frame buffer with non-blocking store and forward
g) 802.1q VLAN support for 64 groups from 4096 IDs
h) 802.1w Rapid Spanning Tree Protocol (RSTP)
i) Very small size of only 10 cm x 8 cm

4) **SDN Controller: A** HP Spectre 360 laptop having 8 GB RAM and 256 GB SSD with Intel CORE i5 processor has been used with Ubuntu 16.05 VM hosted to install OpenDaylight controller. ODL is an open source platform for programmable software defined networks [49]. ODL can be used in many ways in terms of its southbound connections (the underlying network) and its northbound connections (the connection to other platforms such as Docker and OpenStack).

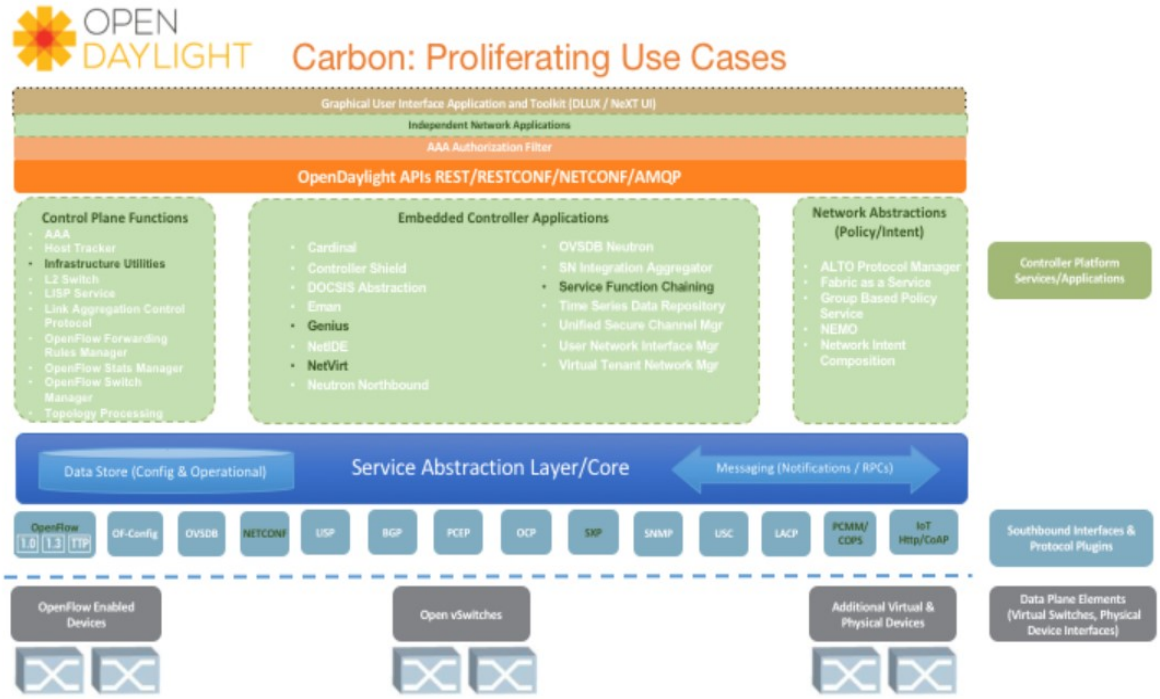

Figure 3.14 OpenDaylight (Carbon) SDN Controller [49]

Figure 3.14 shows the Carbon release of ODL. ODL is a fundamental part of SDNs, where it provides a Cloud-based controller that is able to function where needed while still being open source. This allows for developers to continue working with it, and finding out the full potential of ODL in the world of SDNs.

3.6 Device Discovery Module

The device discovery module works in two phases. In the first phase, it scans all its interfaces and updates the backend repository about the available interfaces for receiving the connection request form IoT devices. The second phase is the device discovery where inputs are the requests coming from IoT devices for new connection or end existing

connection. Each phase of the device discovery module is explained in the following sections.

3.6.1 Phase I: Network Scan

Raspberry Pi 3 is used as an IoT Gateway. In the first phase of discovery module, the Gateway scans all its interfaces, i.e., USB1 for USB to serial connection, Wi-Fi interface, and BLE interface for wireless connections. Table 3.2 describes the notations used in the network scanning algorithm. I is the set of heterogenous interfaces on IoT Gateway, $I_{Selected}$ is the selected interface from I and I_{Active} is the set of active interfaces, where $I_{Selected} \in I$ and $I_{Active} \subset I_{Selected}$

The inputs to the network scanning are a list of all interfaces on Raspberry Pi 3 acting as IoT Gateway. The algorithm for network interface scanning is shown in Algorithm 1.

Table 3.2 Notations used in scanning algorithm for heterogeneous Network Interfaces

Notation	DESCRIPTION
I	Set of Heterogeneous Network Interfaces
$I_{Selected}$	Selected Interface
I_{Active}	Set of Active Interfaces

As shown in Algorithm 1, the device discovery module selects an interface $I_{Selected}$ from the set of all interfaces I. It then scans the interface and if found active then it is added in the set of active interfaces, i.e., $I_{Active.}$

Algorithm 1: Scanning of Heterogeneous Network Interfaces

Input: Set of Heterogeneous Network Interfaces I, Set Selected Interface $I_{Selected}$

Output: Active Interfaces I_{Active},

```
1 for each interface I_Selected ∈ I do
2     Scan I_Selected Interface
3     if I_Selected is Active then
4         Add I_Selected to I_Active
5     end if
6 end for
```

The IoT Gateway Middleware is installed on Raspberry Pi 3 which supports Wi-Fi, Bluetooth and Serial interfaces. The Algorithm 1 scans these network interfaces and keeps a record of their current status. Raspberry Pi 3 can have more interfaces using external adapters and in case of multiple interfaces also the Algorithm 1 works same.

3.6.2 Phase II: Node Detection and Removal Algorithm

In this phase, the inputs to the discovery module are the requests coming from the IoT devices. The requests are of two types: (1) new connection request and (2) connection termination request. Table 3.3 describes the notations used in the node detection and removal algorithm.

Table 3.3 Notations used in New Node Detection and Removal algorithm for heterogeneous Network Interfaces

Notation	DESCRIPTION
L_{active}	Set of active device and link ID
$L_{Wi\text{-}Fi}$	Set active Wi-Fi connected devices

$L_{Bletooth,}$	Set active Bluetooth connected devices
$L_{Serial,}$	Set active Serially connected devices
d	Selected device from L_{active}

Algorithm 2: Discover and Connect Heterogeneous IoT Devices

Input: L_{active}, d is the selected device from L_{active}

Output: $L_{Wi\text{-}Fi}$, $L_{Bletooth}$, L_{Serial}

```
1  for each device connection request
2      Authenticate and connect
3      Add link ID attribute to device ID
4      Update device ID into set L_active
5  end for
6  for each d ∈ L_active
7      if link ID = Wi-Fi then
8          add d to L_Wi-Fi
9      elseif link ID = Bluetooth then
10         add d to L_Bluetooth
11     else
12         add d to L_serial
13     end if
14 end for
```

As shown in Algorithm 2, all the active interfaces of IoT Gateway listens for any new connection requests or termination requests of existing connections. In the first request type, a new connection request is received by the IoT Gateway from an IoT device. The IoT Gateway checks the authorization of the device and if the device meets the authorization criteria, then the Gateway connects the device and updates the device ID and corresponding link ID in the set of active connections. The link ID is stored with an

attribute defining the type of interface it is connected to. Example sensor JY901 connected serially is stored with SN1 as device ID and SL1 as serial link ID.

For example, if a Wi-Fi enabled IoT device sends a connection request to the IoT Gateway, then the IoT Gateway first checks if the device is authorized to establish a connection. If it meets the authorization criteria, then the IoT Gateway will connect the device; otherwise, the connection request will get refused. Once connection established, the IoT Gateway updates the device ID and the link ID in the set of active connections. Along with link ID it stores link attribute as Wi-Fi. Similarly, link attributes as BLE and Serial get updated to corresponding link IDs of BLE and Serial connection, respectively. This link attribute is used to determine the type of device based on the interface connected to. Finally, the device discovery module sends a node detection update to the IoT Middleware repository. The IoT Middleware is hosted on the IoT Gateway and acts as a IoT broker. The applications subscribed to this broker receives an update on new node detection.

In the second request type, a connection termination request is received by the IoT Gateway from the connected IoT device. The IoT Gateway terminates the connection and updates the set of active connections by removing node ID and corresponding link ID of the terminated device. As soon as link ID is removed from the set of active connections the related link attribute is also gets removed. Finally, the device discovery module sends a node removal detection update to the IoT Middleware repository. The applications subscribed to this IoT Middleware broker receives an update on node removal detection. The algorithm for connection termination is presented in Algorithm 3.

Algorithm 3: Terminate Device Connection

Input: L_{active}

```
1   If connected device request disconnection then
2       Check device connection ID in L_active
3       Close the device client connection
4       Remove client ID from the L_active
5   end if
6   if connected device no more sending data then
7       wait for time T
8       if no response from the device then
9           Close the device client connection
10          Remove client ID from the L_active
11      end if
12  end if
```

3.7 Chapter Summary

IoT networks suffer from huge volume of data on the network, hence IoT can take the advantages of the SDN technologies for efficient device and traffic management. A centralized and distributed SDN managed IoT network architecture was proposed for efficient management of the devices and application services (see Section 3.1 and 3.2).

Section 3.3 discussed the system architecture for SDN managed IoT networks. To realize the IoT network a prototype of SDN managed IoT network testbed was designed for conducted experiments to evaluate performance analysis of the proposed Middleware solution.

The proposed Middleware deployed on Edge network (IoT Gateway device) runs a device discovery module to scan and connect all the devices in Wi-Fi, Bluetooth and Serial

network of the IoT Gateway. The connected devices publish their ID and attributes to the Middleware broker which notifies subscribed applications on connection of any new device node or removal of any device node from the IoT network.

Chapter 4

Result Analysis and Discussion

This chapter presents performance evaluation of the proposed Middleware solution in SDN managed IoT network using benchmark tools. The experiments are conducted on the testbed comprising hardware resources as presented in Section 3.4. The experimental setup and the approach to the experiments conducted for performance evaluation of the proposed Middleware solution is presented in Section 4.1. The experiments are conducted to evaluate system performance, application performance and network performance of the proposed Middleware solution. Section 4.2 presents the results and analysis of system performance tests conducted using the proposed Middleware solution on the IoT Gateway Edge device. Section 4.3 shows the results of application based performance tests with varying publication frequencies and subscription requests at the Middleware Broker. As discussed before, it is beneficial to deploy a Middleware solution close to the Edge network for low latency applications. In Section 4.4, the proposed Middleware solution is analyzed in terms of network performance by deploying it on the IoT Gateway as an Edge device versus on the Microsoft Azure Cloud. The conclusive analysis of the experiments presented in the chapter is summarized in Section 4.5

4.1 Experimental Setup and Methodology

A set of benchmark tools are used to evaluate the performance of the proposed Middleware solution deployed on the IoT Gateway. The experiments are conducted on

the prototype testbed designed for a SDN-based IoT network. The experiments for performance evaluation are categorized in three types as follows:

1) **System performance of IoT Gateway with Middleware**

For evaluating system performance, two different setups are used. In the first setup, the IoT Gateway runs device discovery module without the Middleware, while in second setup, the IoT Gateway runs the same device discovery module with the Middleware deployed on the IoT Gateway. Experiments are conducted with varying the number and the type of IoT devices during device discovery. These experiments explore the potential of deploying Middleware on a resource constrained IoT Gateway (Raspberry Pi 3) acting as the Edge device.

2) **Application performance of the Middleware solution**

The Middleware application performance tests are conducted on the proposed Middleware solution hosted on IoT Gateway, i.e., close to the user Edge versus the Middleware hosted on the Microsoft Azure Cloud. Large datasets are published as individual data elements to test the publish times for a number of requests. The experiments are conducted by varying the publication frequency and subscription requests at the Middleware Broker.

3) **Network performance of the Middleware solution**

New node discovery and node removal discovery applications are used for evaluating network performance of the Middleware solution deployed on the IoT Gateway versus

the Microsoft Azure Cloud. The experiments are conducted to evaluate the device discovery time by the subscriber application hosted on IoT devices.

4.2 System Performance of IoT Gateway with Middleware

The following subsections presents the experimental results of the IoT Gateway performance with the Middleware solution using benchmark tools presented in Table 4.1. The IoT Gateway is running the device discovery module for performance evaluation of the IoT Gateway with and without Middleware deployed.

Table 4.1. Benchmark tools adopted for testing system performance

Resource	Benchmark Tool	Performance Metric
CPU	sysbench [60]	Seconds
CPU	mpstat [66]	Percent of usage
Memory	mbw [61]	MiB/s
Power	PowerTop [63]	W (Watt)

It should be noted that the benchmark tools can have some intrinsic limitations, hence the following results should be viewed critically. However, we can still deduce valuable implications and discover insightful patterns from the results.

4.2.1 CPU Performance

For evaluating CPU utilization, *sysbench* 0.4.12 is used. *sysbench* is a modular, cross-platform, and multithreaded benchmark tool for evaluating the system performance. It

executes a stress test designed to challenge the CPU by performing device discovery. The performance metric is the *execution time* (measured in seconds) and a lower execution time implies better performance. Experiments are conducted to measure the average execution time to discover 4 IoT devices and discover removal of 4 IoT devices (list the type of IoT devices here) by the IoT Gateway. Figure 4.1 shows the result of CPU Performance for an average of 10 tests conducted.

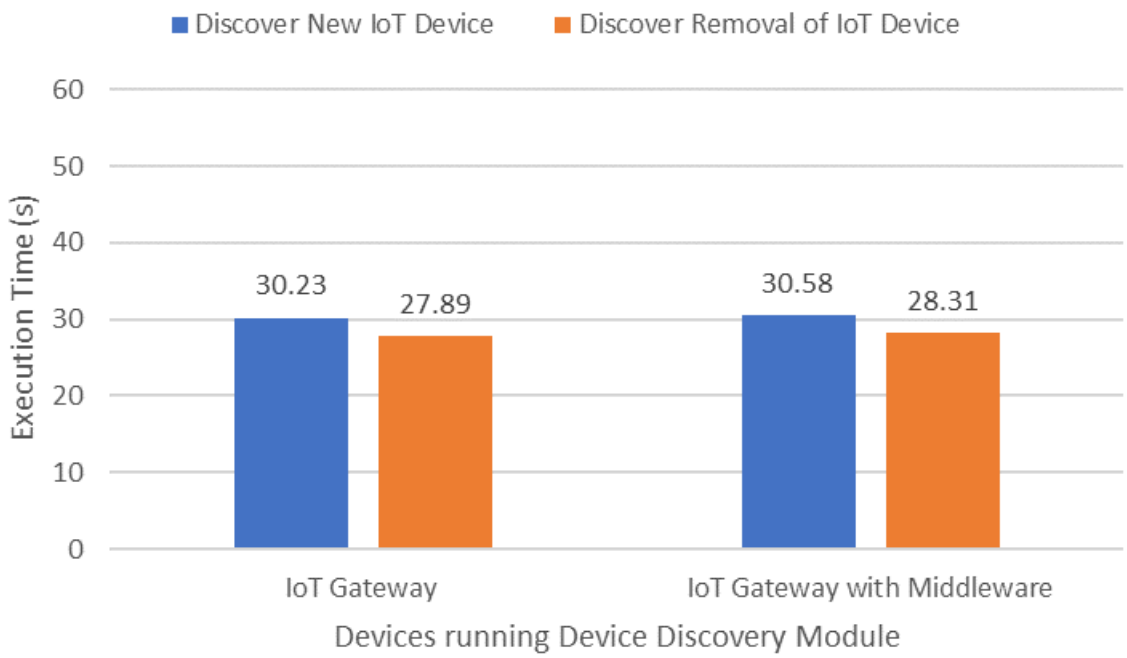

Figure 4.1 CPU Performance

It can be observed from Figure 4.1 that the execution time of the Device Discovery Module by IoT Gateway with the Middleware is 30.58 seconds which is higher than the execution time by the IoT Gateway without the Middleware. The Middleware solution has a Middleware data repository where the devices status and attributes are updated

continuously to keep a track by the application services. This introduces few more database transactions in the Middleware and thus higher execution time.

Table 4.2 Results of Execution time for CPU performance

		Mean (seconds)	Standard Deviation	Confidence Interval (95%)
Discover New IoT Device	**IoT Gateway**	30.23	2.83	30.23 ± 0.88
	IoT Gateway with Middleware	30.58	2.78	30.58 ± 0.83
Discover Removal of IoT Device	**IoT Gateway**	27.89	2.24	27.89 ± 0.68
	IoT Gateway with Middleware	28.31	2.41	28.31 ± 0.78

Although the execution time of IoT Gateway with the Middleware is slightly higher, but it can also be observed from Table 4.2 that the difference of mean execution time between both the Gateway devices is only 0.35 seconds during discovery of new IoT device and the IoT Gateway with Middleware performs better in terms of low standard deviation and confidence interval. While during discovery of removal of an IoT Device the IoT Gateway without Middleware performs slightly better but the differences are very low. Thus, we can conclude that deploying the Middleware solution does not introduce more overload on CPU performance, but the CPU utilization could get increased as applications get more CPU intensive.

It can also be observed that the execution time for the discovery of the removal of the IoT devices in both the setups is relatively less than that of discovering of new devices. This is because the removal process excludes the authorization of devices which is needed by new device discovery.

Figure 4.2 shows the CPU utilization of the device discovery module in detail using command line utility called *mpstat* [60] when 4 IoT devices are discovered by both the IoT Gateway and the IoT Gateway with Middleware.

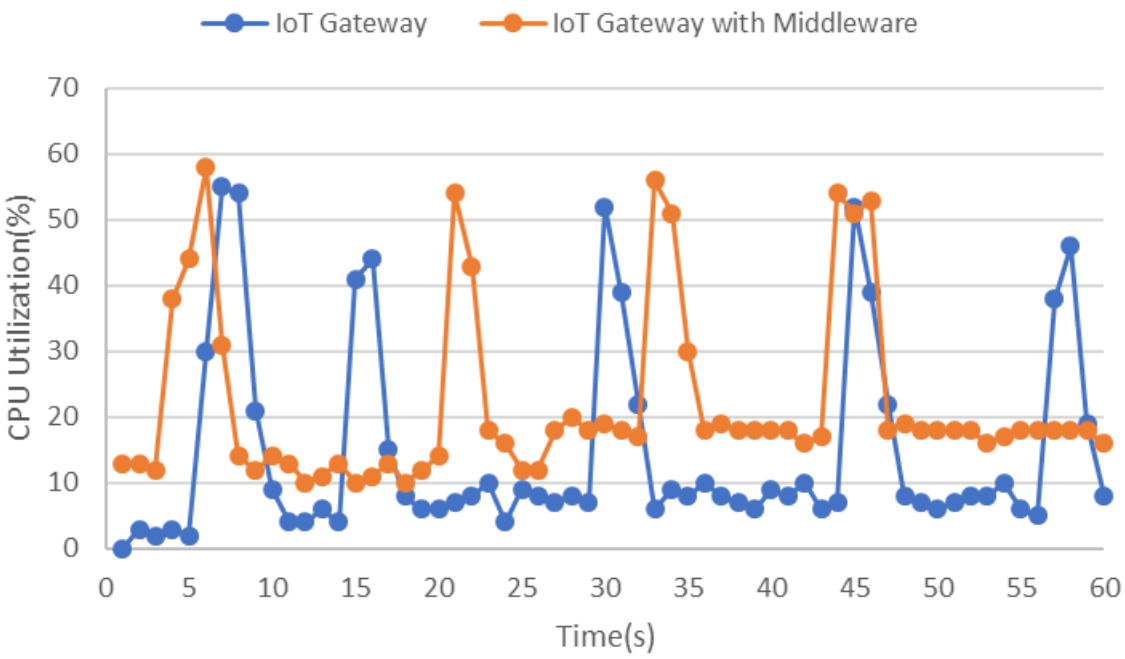

Figure 4.2 CPU utilization during device discovery on IoT Gateway with and without Middleware

When the Middleware is deployed on the IoT Gateway, the CPU usage is generally around 14%-16%. It also shows some sudden spikes in CPU utilization, this is because more CPU is utilized during the discovery of every new device. Device discovery includes receiving a new connection request, authorizing the device, establishing the connection and updating repository with the new node ID and Link ID with attributes. The highest CPU utilization reaches up to57% with the device the discovery module, whereas when Middleware is also deployed on the IoT Gateway the CPU utilization reaches a maximum

of 59%. This concludes that deploying the proposed IoT Middleware on the IoT Gateway does not impose much overhead and we believe that this is reasonable percent of resource utilization suitable for resource-constrained IoT devices.

4.2.2 Memory Performance

The benchmark tool *mbw* (memory bandwidth) [61], measured in MiB/s, is used for evaluating memory performance. It determines the available memory bandwidth by copying large arrays of data into memory. It uses three different methods, namely MEMCPY, DUMB and MCBLOCK. One of the advantages of this tool is that it is neither tuned to extremes, nor is it aware of hardware architecture, which models the behavior of real applications. To ensure accuracy, the swap feature has been turned off and caches have been cleared before each round of experiment.

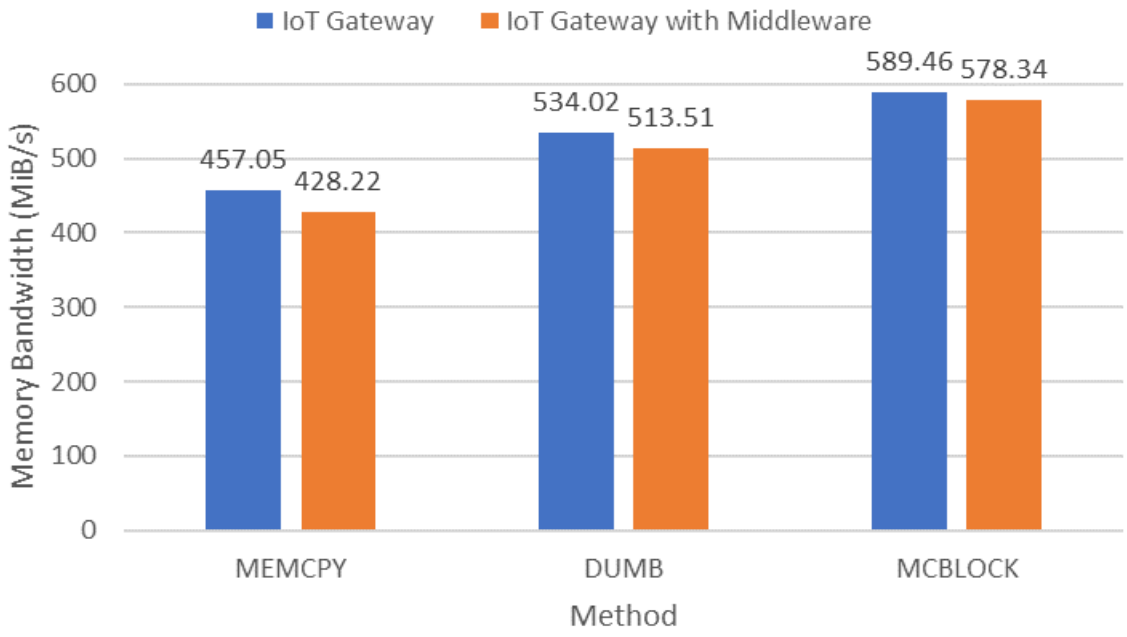

Figure 4.3 Memory Performance test during device discovery on IoT Gateway with and without Middleware

Figure 4.3 shows the memory bandwidth of MEMCPY, DUMB and MCBLOCK tests while running the device discovery module in the background.

Table 4.3 Results of Memory Performance

		Mean (MiB/s)	Standard Deviation	Confidence Interval (95%)
MEMCPY	**IoT Gateway**	457.05	21.34	457.05± 15.62
	IoT Gateway with Middleware	428.22	23.56	428.22 ± 15.67
DUMB	**IoT Gateway**	534.02	26.93	534.02 ± 12.23
	IoT Gateway with Middleware	513.51	28.32	513.51 ± 12.98
MCBLOCK	**IoT Gateway**	589.46	34.35	589.46 ± 22.79
	IoT Gateway with Middleware	578.34	33.98	578.34 ± 21.20

In Table 4.3 results of memory performance is presented in terms of mean, standard deviation and confidence interval (95%). From all the three methods used for testing, it can be noticed that the bandwidth of the IoT Gateway with Middleware is relatively less than that of the IoT Gateway without Middleware. The difference is quite small in comparison and it can be concluded from the performance, the proposed Middleware does not overburden the IoT Gateway.

4.2.3 Power Consumption

This test is executed using benchmark *PowerTop* [63], a Unix based command line utility for measuring power consumption, during performance evaluation of device discovery application. The tests are conducted on Raspberry Pi which acts as the IoT Gateway with two setups. Firstly, when Raspberry Pi 3 is acting only as the IoT Gateway and secondly

when Raspberry Pi 3 is acting as the IoT Gateway with Middleware. For relative comparison the power consumption on Raspberry Pi 3 was determined without any application running. Figure 4.4 illustrates the results of performing device discovery scan with an increasing number of devices. For device scan, the first experiment includes 3 devices (3 Wi-Fi devices), the second experiment includes 5 devices (1 Bluetooth and 4 Wi-Fi devices) and the third experiment includes 5 devices (2 Serial and 3 Wi-Fi devices).

Typically, Raspberry Pi 3 uses 1.4 Watts to perform the system operations. From Figure 4.4, it can be observed that there is not much difference in the power consumption in both of the setups. The IoT Gateway with Middleware setup consumes relatively little more power as it also runs a database and the Broker applications. Though the average power consumption is around 2.6-2.8 Watts, it can get increased when the publication frequency or the number of subscription requests increases at the Middleware Broker.

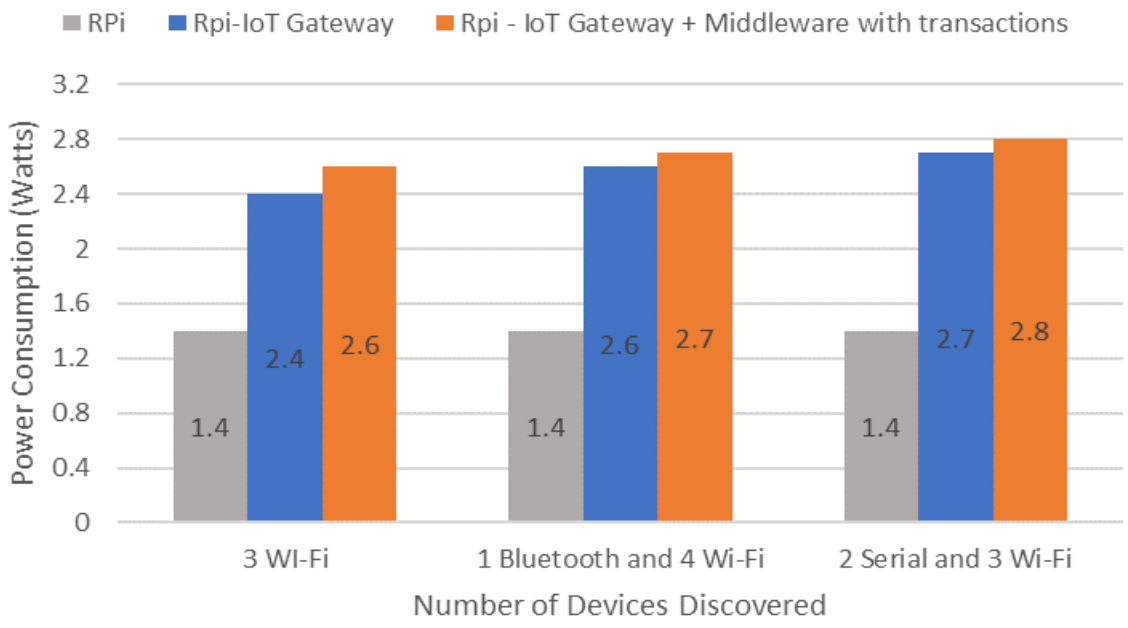

Figure 4.4 Power Consumption test during device discovery on IoT Gateway with and without Middleware

4.3 Middleware Application Performance

A set of experiments are conducted to test the interoperability between IoT devices and applications and to measure the matching time by the Middleware Broker by changing the number of publications and subscriptions. Experiments are also conducted for system performance evaluation with different application requests.

4.3.1 Test for Interoperability

Interoperability mainly deals with different types of systems or components to exchange information and to use the information that has been exchanged. In the context of this thesis, interoperability refers to the operation of different IoT devices to exchange information to different application services using different protocols. The experiment is conducted on IoT devices by installing a MQTT Client on BlackBerry Priv device, AMQP Client on iPhone 7 and MQTT client on JY901 sensor node. Blackberry Priv device is communicating over Bluetooth and iPhone 7 over Wi-Fi network and JY901 sensor node on serial connection. All these devices send telemetry data to RabbitMQ AMQP Broker installed at the IoT Gateway Middleware. A MQTT plugin is installed with RabbitMQ Broker to support IoT device sending data using MQTT data protocol as well. A telemetry dashboard application is designed, which is a AMQP subscriber client to receive data from the IoT Gateway broker. Experiment details are presented in table 4.

Table 4. Experiment for Interoperability between IoT Devices and Application

IoT Device	Publisher Client	IoT Middleware Broker	Subscriber Client
JY901 sensor node	MQTT	AMQP Broker	AMQP Client based dashboard

iPhone 7	AMQP	AMQP Broker	AMQP Client based dashboard
Blackberry Priv	MQTT	AMQP Broker	AMQP Client based dashboard

Figure 4.5 shows the telemetry data transmitted from all the three IoT devices on the dashboard application. Thus, the proposed middleware solution achieved interoperability between three different types of IoT devices, working on MQTT and AMQP protocols and connected through Wi-Fi, Bluetooth and serially, and telemetry dashboard application which is a AMQP subscriber.

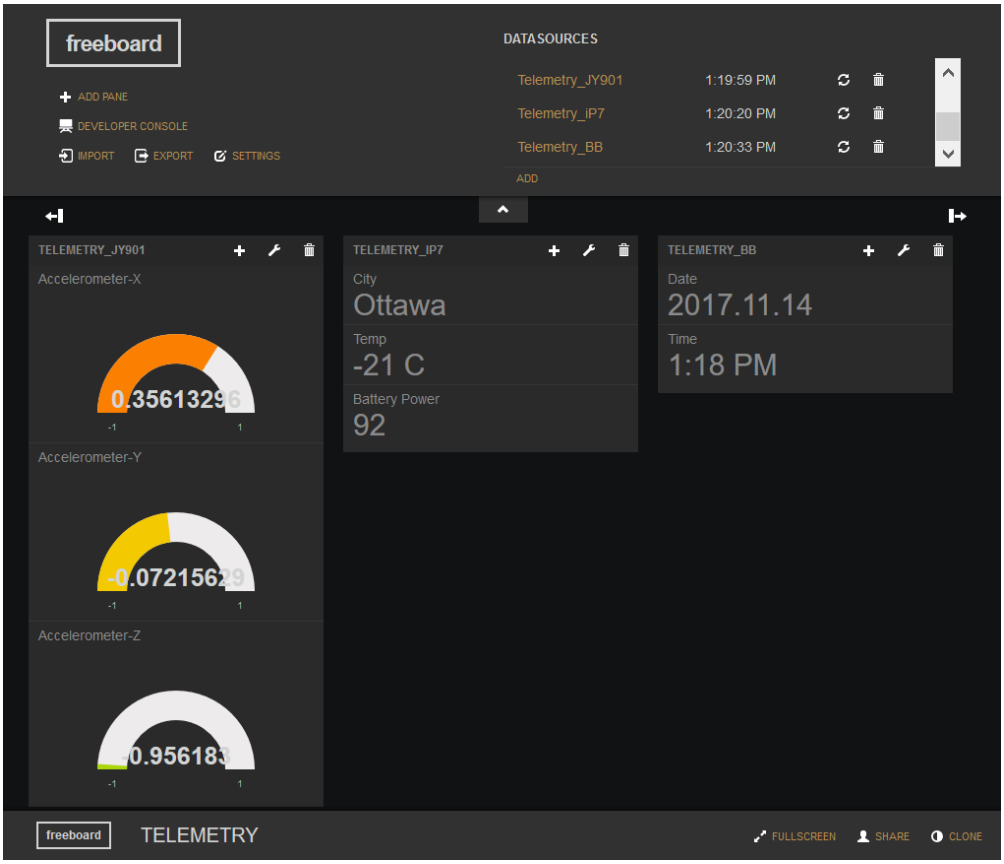

Figure 4.5 Dashboard for telemetry application

4.3.2 Subscription Requests Matching Time by Middleware Broker

Matching time is the time required by a Broker to perform matching between a received subscription and a newly created publication, i.e., the time required to determine if a publication satisfies the constraints of at least one subscription.

JY901 sensor node has a baud rate of 2400 bps to 921600 bps. If a message size is 100 bytes, then the publication rate of sensor can range from 3 messages/s to 1152 messages/s. Since we have 4 publishers (JY901 sensor nodes), hence the total publication rate received by the broker from all publishers can also range from 12 to 4608 messages/s.

The matching time is measured while varying the number of subscriptions received by the Middleware Broker. The number of subscriptions are varied from 20 to 200 while keeping the publication rate of each publisher as 200 messages per second, where a message size is of 100 bytes. In the experiment the number of publishers is fixed to 4, hence the total publish rate received by broker is 800 messages per second.

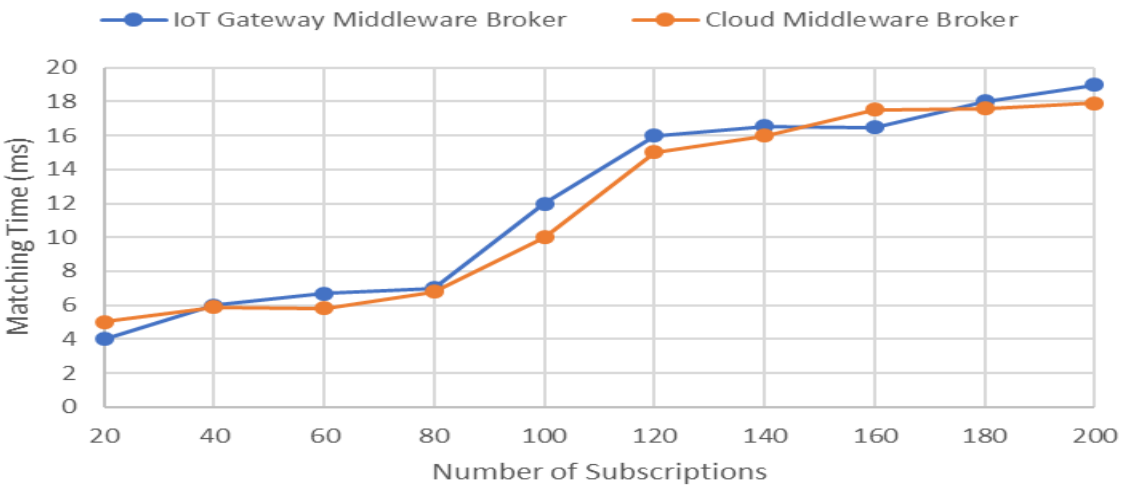

Figure 4.6 Matching Time of Middleware Broker compared to number of subscriptions

From Figure 4.6, it can be observed that the matching time increases as the number of subscriptions increases. Reason for the increasing matching time is due to increased subscription requests for searching and matching published topics. The Middleware Broker at IoT Gateway has lower matching time when a small number of subscriptions is received by the Broker. When a large number of subscriptions is used the difference in performance between the Middleware Brokers at IoT Gateway and Cloud is relatively small.

4.3.3 Publisher Topic Matching Time by Middleware Broker

The matching time was measured while varying the publication rate from 100 messages per second to 1000 messages per second, where the message size is 100 bytes and the subscription rate is at a constant number of 100. The number of publishers is fixed to 4; hence, publish rate received by broker is from 400 message/s to 4000 message/s. In other words, the experiment identifies all satisfied subscriptions for a given number of publication rate.

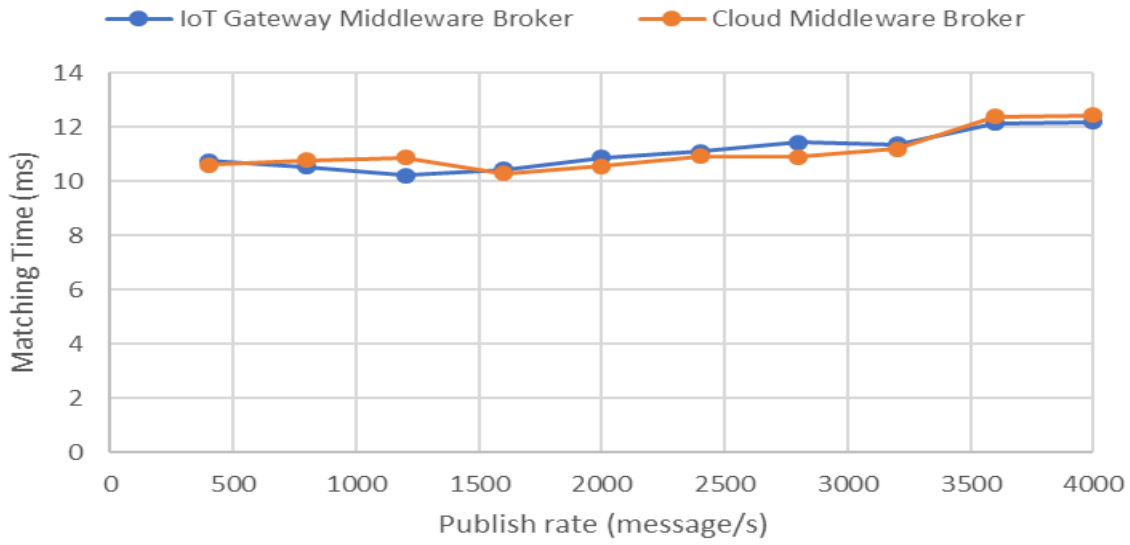

Figure 4.7 Matching Time of Middleware Broker compared to publication rate

Figure 4.7 shows that the matching time is not significantly affected by the publication rate as it depends mostly on the number of stored subscriptions.

4.3.4 Effect of the Number of Subscriptions

The experiment is conducted to evaluate the performance of the Middleware Broker running on the IoT Gateway device in terms of memory and power utilization with increasing the number of subscription requests. Figure 4.8 shows the memory and power utilization by the Middleware Broker when the number of subscription requests increased from 20 to 200; the publication rate is kept at 1000 messages per second where each message size is of 100 bytes.

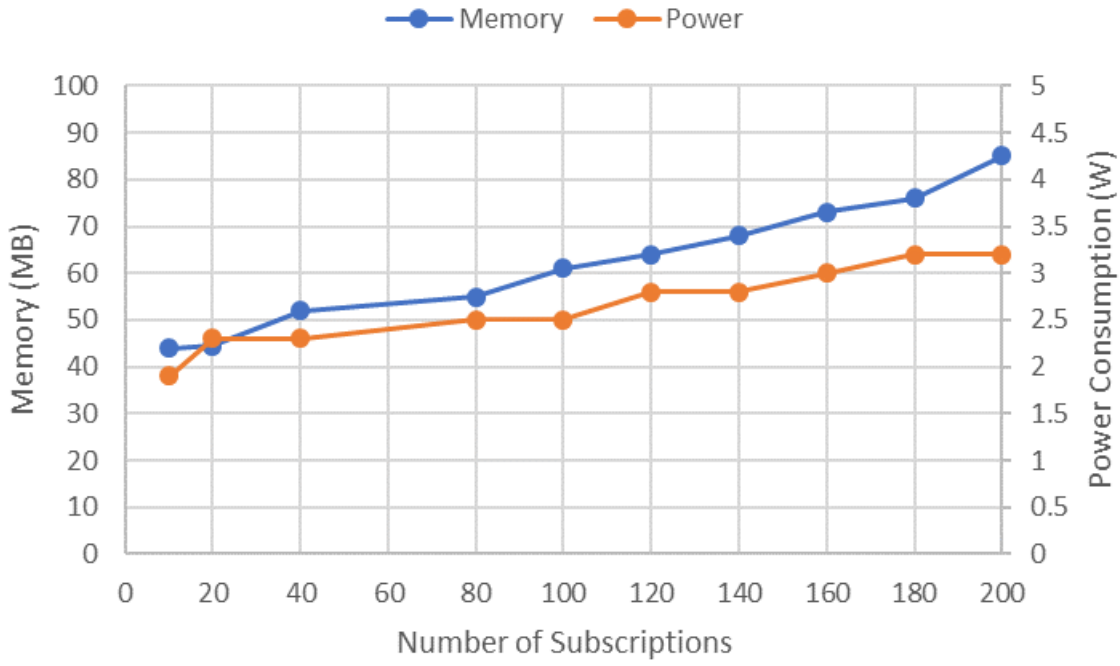

Figure 4.8 Power and Memory Consumption by Middleware Broker compared to number of subscriptions

It can be observed that the power consumption increases with an increase in the number of the subscription requests and lies in between 1.8 W to 3.1 W. The memory usage also shows a similar increasing pattern with an increase in the subscription requests and reaches a maximum of 88 MB. It can be concluded that the proposed Middleware Broker is a lightweight solution and can be easily run on resource constrained IoT devices for a wider application.

4.3.5 Effect of the number of Publications

Another experiment is conducted to evaluate the performance of the Middleware Broker running on the IoT Gateway device by increasing the publication rate and maintaining the subscription request at a constant of 100. Figure 4.9 shows the memory and power consumption by the Middleware Broker when the publication rate is increased from 100 messages/second to 4000 messages/second, where each message size is of 100 bytes.

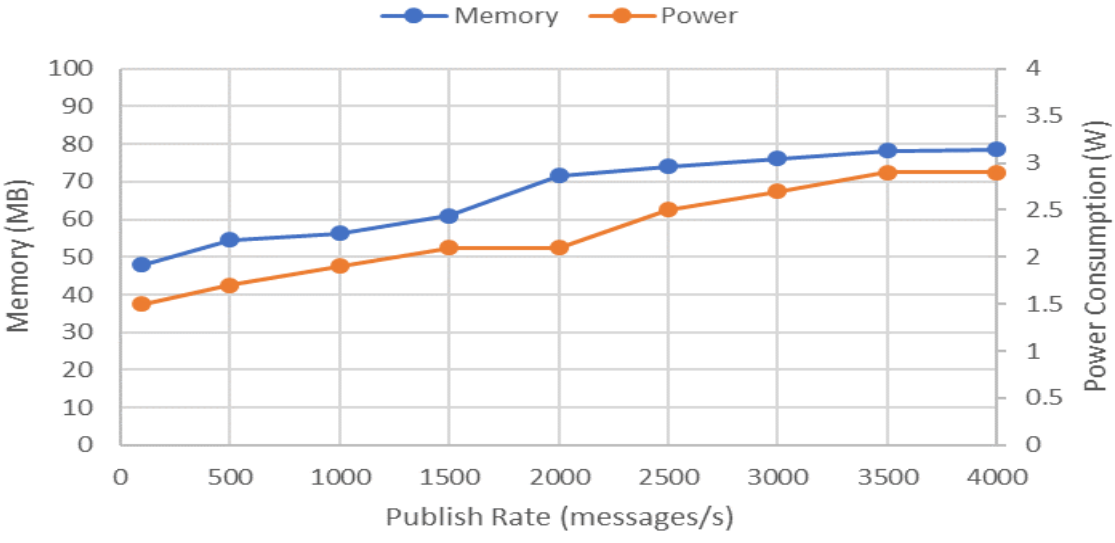

Figure 4.9 Power and Memory Consumption by Middleware Broker compared to publication frequency

In our experiment when publication rate received by the broker is increased to from 100 message/s to 4000 messages/s, the power consumption reaches 2.9 W. Similarly, memory usage also increases to a maximum of 79 MB, as it involves preprocessing of increased data volume and increased database transactions. The maximum power consumption is 2.9 W and a memory usage of 79 MB, which is low and acceptable [64] for Raspberry Pi 3.

4.4 Network Performance

The following subsections present the experiment results of the network performance by the proposed Middleware deployed on an Edge network (a nearby IoT Gateway) versus the proposed Middleware deployed on a remote Cloud network. The Microsoft Azure Cloud is used to host the Middleware Broker on Cloud, which is provided as a service at Canada Central Zone in the Toronto data center, the closest public Cloud from Ottawa area. The experiment is conducted in two setups as explained below:

1) Application client (publisher/subscriber) is running on IoT device (iPhone 7) close to the Edge network and the Middleware Broker deployed on Microsoft Azure Cloud.

2) Application client (publisher/subscriber) is running on IoT device (iPhone 7) close to the Edge network and the Middleware Broker deployed on the IoT Gateway (Raspberry Pi 3).

4.4.1 Round Trip Time (RTT)

Round trip time is the time taken by packets to be transmitted from a source to a destination and received back by the source as an acknowledgement. The *ping* utility has been used to measure the RTT between a sensor node and the Brokers. The test is repeated 10 times and an average RTT is calculated.

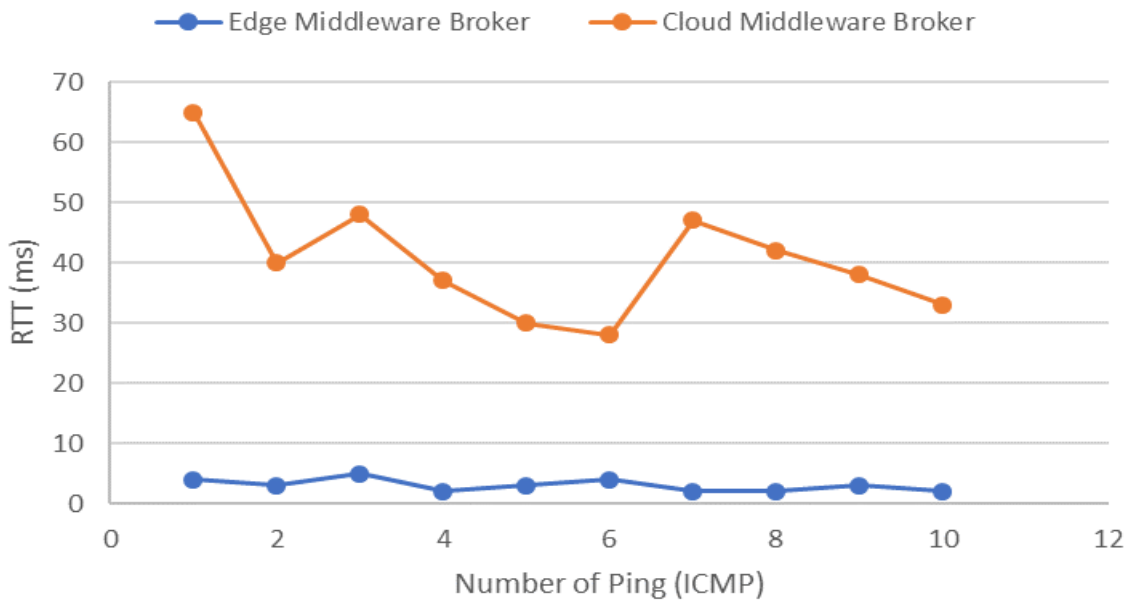

Figure 4.10 RTT of Edge Middleware Broker and Cloud Middleware Broker from sensor node at IoT network

Figure 4.10 shows the RTT between a sensor node at an IoT network and the Edge Middleware Broker versus RTT between a sensor at the IoT network and the Cloud Middleware Broker (Microsoft Azure Cloud). The average RTT for the Cloud Middleware Broker is 40.8 ms, but only 3.4 ms for the Edge Middleware Broker. As expected it can be observed that the RTT of the Cloud Broker is very high compared to that of the Edge Broker. The delay could have been more if we would have chosen a Cloud service location

farther from the Canada Central zone. Further, the actual location of cloud services might be tentatively migrated to another location by the Cloud Service Provider due to some reason. The results clearly show an advantage of deploying the Middleware Broker on the IoT Gateway as an Edge device., which is important for time critical IoT applications.

4.4.2 New Node Discovery

This experiment is conducted to measure the time taken by the applications (subscribers) to receive a notification of the new node discovered in the IoT network from the Edge Middleware Broker and a Cloud Middleware Broker. Figure 4.11 shows the experiment results conducted 10 times during network traffic peak hours i.e. between 9:00 AM to 11:00 AM.

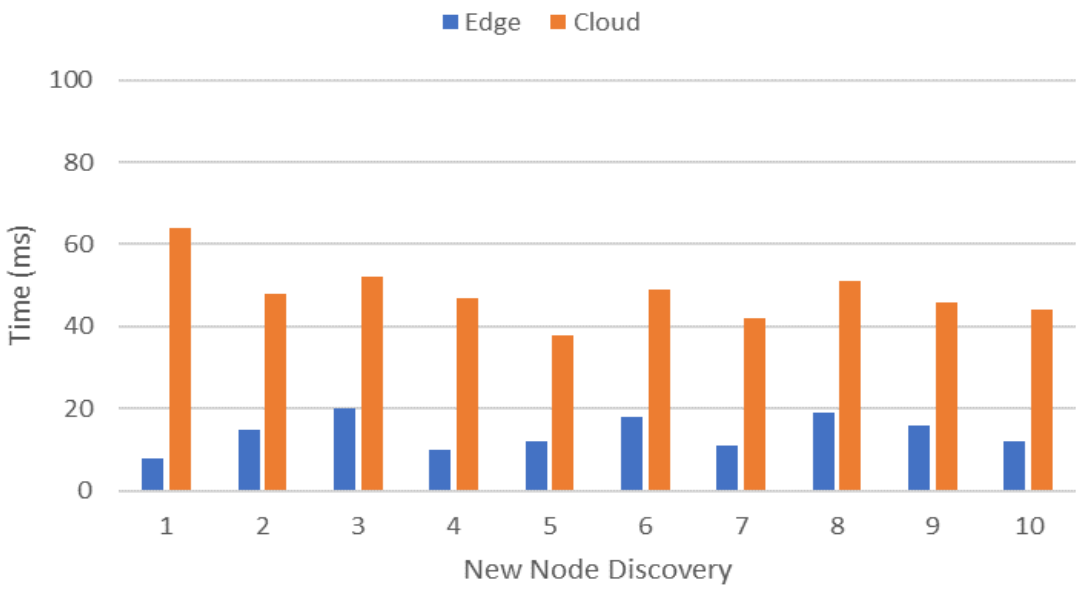

Figure 4.11 Time taken to discover a new IoT device

It can be observed from the graph that the average time taken to discover a new node is about 48.2 ms and a maximum time of 64.3 ms by the Cloud Middleware Broker,

while the Edge Middleware Broker takes an average of 12.6 ms and a maximum of 19.2 ms to discover a new node in the IoT network. The vast time difference accounts for the network latency to receive device notification by a subscriber application. The subscriber application running on IoT devices at an edge can take advantage of deploying the Middleware broker at the Edge network like IoT Gateway to receive subscribed data with low latency. This approach is most suitable for latency sensitive IoT applications.

4.4.3 Node Removal Detection in Network

Similarly, another experiment is conducted to measure the time taken by the applications (subscribers) to receive a notification of the removal of a node from the IoT network from an Edge Middleware Broker and a Cloud Middleware Broker. Figure 4.12 shows the experiment results conducted 10 times during network traffic peak hours, i.e., between 9:00 AM to 11:00 AM.

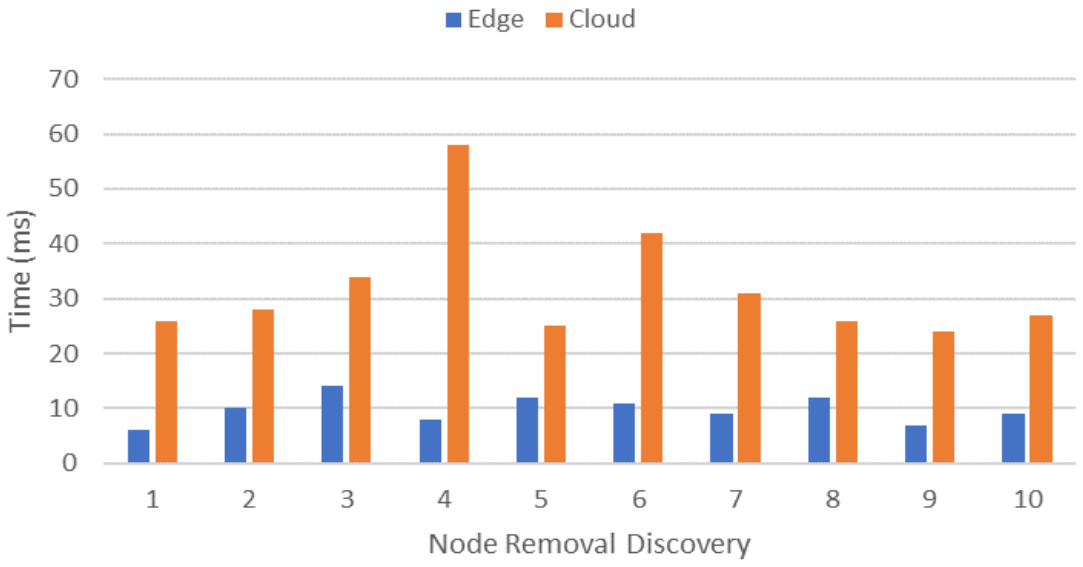

Figure 4.12 Time taken to discover removal of IoT device

It can be observed from the graph that the average time taken to discover the removal of a node is about 33.6 ms and a maximum time of 58.4 ms by the Cloud Middleware Broker, while the Edge Middleware Broker takes an average of 9.8 ms and a maximum of 14.1 ms to discover the removal of a node from the IoT network. The variation of delay for the Cloud Middleware Broker is also high.

4.5 Chapter Summary

In this chapter, a performance analysis of the proposed Middleware Broker solution was presented by conducting experiments on the designed SDN managed IoT network testbed. Firstly, the experiments were conducted to evaluate system performance when the proposed Middleware Broker solution is deployed on the IoT Gateway (Raspberry Pi 3). The results showed that the proposed Middleware was a lightweight solution which is suitable for the resource constrained devices, which is a prominent requirement of IoT networks.

Secondly, the Middleware application performance was evaluated by matching published topics with the requests and varying publishing frequency and subscription requests. It was found that the matching time by both the Middleware Brokers deployed on the Edge network (IoT Gateway) and the Cloud network (Microsoft Azure) performed similarly. The resource consumption by the Middleware Broker on the IoT Gateway is found to be in reasonable limits.

Thirdly, the network performance was evaluated between the Cloud Middleware Broker and the Edge Middleware Broker by conducting device discovery by the subscribing application. It was found that the proposed Middleware solution performed

efficiently on the Edge network and supported interoperability between different IoT devices and applications.

Chapter 5

Conclusions and Future work

In this chapter, section 5.1 provides a summary on the approach adopted by this thesis and the major findings. Section 5.2 points out some areas for future work.

5.1 Conclusions

This thesis focuses on the development of an IoT Middleware solution which can provide a horizontal data integration for diverse IoT devices and provide interoperability between diverse applications and devices by abstracting and translating protocols and standards. The objective is to design a Middleware which could be used to deploy both on IoT Gateways, i.e., closer to the user Edge and on a Cloud network as per the application requirement. Majority of the Middleware solutions are developed for the Cloud network which works on IP technology. To deploy a Middleware on IoT gateway it must support multiple protocols not only for applications, but also for underlying diverse IoT devices and protocols. Deploying Middleware either on the Cloud or on an IoT gateway has both benefits and drawbacks. This thesis researches on developing a Middleware solution and explores the potential of Middleware deployed closed to the user Edge.

To evaluate the performance of the designed Middleware solution, a SDN based IoT network testbed has been designed and developed using sensors, Raspberry Pi 3, SDN switches, the OpenDaylight SDN controller and the Microsoft Azure Cloud services. The proposed Middleware is deployed on the IoT Gateway and on the Microsoft Azure Cloud for performance evaluation with respect to CPU performance, memory performance,

network throughput, power consumption, publication message matching time, application execution time and latency. The experimental results show that the proposed Middleware solution provided interoperability between heterogenous IoT devices (including JY901 sensor nodes, iPhone and Blackberry phone) transmitting data on heterogenous network (W-Fi, Bluetooth and Serial) using different data protocols (MQTT and AMQP) and subscriber applications (AMQP subscriber). The experiments based on system performance, application performance and network performance shows that the proposed Middleware solution could be deployed on IoT Gateway, i.e., in close proximity to the user Edge which could provide better local data analysis for time critical applications in real time. The proposed Middleware solution is a lightweight solution, and the experiments shows that it is suitable to be deployed on resource constrained single board computers like Raspberry Pi 3 for IoT applications.

5.2 Future Work

The Middleware technology solutions for IoT are still under active development. As a contribution to the development, the main objective of this thesis is to design a Middleware solution which can be deployed close to the user Edge, i.e., on IoT Gateways for the applications where the latency is critical. In future directions this thesis could be extended to implementing lightweight virtualization techniques which could provide high versatility and customizability to the Middleware platform. The Docker container technology [65] could be utilized in order to customize the IoT platform by offering data processing services.

Further a configuration module could be developed as a part of the Middleware solution which can be used to configure different categories of the IoT devices in the IoT network.

Configuration details and sensing strategies such as scheduling, sampling rate, data acquisition method, and protocols can be designed for each individual sensor by the higher-level devices and thus can be pushed towards the lower layers.

Future research direction can also focus on providing IoT security through the Edge Middleware solution, which is one of the major challenges for IoT networks.

The IoT communication protocols and standards are still evolving. Hence, it will be worthwhile to investigate on other protocols to be included in the proposed Middleware solution in the future.

References

[1] ITU-T Y.2060: Overview of the Internet of Things, https://www.itu.int/ITU-T/recommendations/rec.aspx?rec=y.2060. Last accessed on Jan. 1st, 2018.

[2] IoT's Challenges and Opportunities, https://www.gartner.com/technology /research/ internet-of-things/report. Last accessed on Jan. 4th, 2018.

[3] T. Dillon, C. Wu and E. Chang, "Cloud Computing: Issues and Challenges," *Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications*, 2010, pp. 27-33.

[4] N. M. Gonzalez, "Fog computing: Data analytics and Cloud distributed processing on the network edges," *Proceedings of the 35th International Conference of the Chilean Computer Science Society (SCCC)*, 2016, pp. 1-9.

[5] Network Softwarization, https://www.ietf.org/lib/dt/documents/LIAISON/liaison-2016-02-26-itu-t-sg-13-ietf-ls-on-report-on-standard-gap-analysis-from-itu-t-focus-group-on-imt-2020-and-on-extension-of-lifetime-of-focus-g-attachment-2.pdf. Last accessed on Jan. 6th, 2018.

[6] M. Elkhodr, S. Shahrestani and H. Cheung, "The internet of things: new interoperability, management and security challenges," *International Journal of Network Security & Its Applications (IJNSA)*, vol.8, no.2, 2016, pp. 85-102

[7] S. Abdelwahab, B. Hamdaoui, M. Guizani and T. Znati, "Network function virtualization in 5G," *IEEE Communications Magazine*, vol.54, no.4, 2016, pp. 84-91

[8] ETSI NFV MANO, http://www.etsi.org/deliver/etsi_gs/NFV MAN/001_099/001/01.01.01_60/gs_NFV-MAN001v010101p.pdf. Last accessed on Jan. 6th, 2018.

[9] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," *Proceedings of the IEEE*, vol. 103, no. 1, 2015, pp. 14-76.

[10] J. Tourrilhes, P. Sharma, S. Banerjee and J. Pettit, "SDN and OpenFlow Evolution: A Standards Perspective," *Computer*, vol. 47, no. 11, 2014, pp. 22-29.

[11] SDN Architecture, https://www.sdxcentral.com/sdn/definitions/inside-sdn-architecture/. Last accessed on Jan. 6th, 2018.

[12] NFV, https://portal.etsi.org/Portals/TBpages/NFV/Docs/NFV_White_ Paper3 .pdf. Last accessed on Jan. 6th, 2018.

[13] C. Mouradian, D. Naboulsi, S. Yangui, R. H. Glitho, M. J. Morrow and P. A. Polakos, "A Comprehensive Survey on Fog Computing: State-of-the-art and Research Challenges," *IEEE Communications Surveys & Tutorials*, vol. PP, no. 99, 2017, pp. 1-1.

[14] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli," Fog computing and its role in the internet of things," *Proceedings of the 1st ACM Workshop on Mobile* Cloud Computing (MCC), 2012, pp. 13-16.

[15] I. Stojmenovic and S. Wen, "The Fog Computing Paradigm: Scenarios and Security Issues," *Proceedings of the Federated Conference on Computer Science and Information Systems*, 2014, pp. 1–8

[16] M. Firdhous, O. Ghazali and S. Hassan “Fog Computing: Will it be the Future of Cloud Computing?,” *Proceedings of the 3rd International Conference on Informatics & Applications*, 2014

[17] Internet of Things Global Standards Initiative, https://www.itu.int/en/ITU-T/gsi/iot/Pages/default.aspx, (last accessed on Jan. 6th, 2018).

[18] IoT communication stack, https://www.postscapes.com/internet-of-things-protocols/. Last accessed on Jan. 6th, 2018.

[19] The Internet of Things: New Interoperability, Management and Security Challenges, https://arxiv.org/ftp/arxiv/papers/1604 /1604.04824.pdf. Last accessed on Jan. 6th, 2018.

[20] M. Welsh and G. Mainland, "Programming Sensor Networks Using Abstract Regions," *Proceedings of the NSDI*, 2004, pp. 3-3.

[21] Y.-K. Chen, "Challenges and opportunities of internet of things," *Proceedings of the 17th Asia and South Pacific Design Automation Conference (ASP-DAC),* 2012, pp. 383-388.

[22] C. C. Aggarwal, N. Ashish, and A. Sheth, "The internet of things: A survey from the data-centric perspective," *Managing and mining sensor data*: *Springer*, 2013

[23] N. A. Ali and M. Abu-Elkheir, "Data management for the internet of things: Green directions," *Proceedings of the Globecom Workshops (GC Workshops)*, 2012, pp. 386-390.

[24] L. Yang, S. Yang, and L. Plotnick, "How the internet of things technology enhances emergency response operations," *Technological Forecasting and Social Change*, vol. 80, no. 9, 2013, pp. 1854-1867.

[25] M. P. Papazoglou and D. Georgakopoulos, “Introduction: Service oriented computing,” *Communication ACM*, vol. 46, no. 10, 2003, pp. 24–28.

[26] A. H. Ngu, M. Gutierrez, V. Metsis, S. Nepal and Q. Z. Sheng, "IoT Middleware: A Survey on Issues and Enabling Technologies," *Internet of Things Journal*, vol. 4, no. 1, 2017, pp. 1-20.

[27] The TerraSwarm Research Center, 2013, https://terraswarm.org. Last accessed on Jan. 6th, 2018.

[28] Nest, 2014, https://developer.nest.com. Last accessed on Jan. 6th, 2018.

[29] D. Raggett, “The Web of Things: Challenges and Opportunities,” *IEEE Computer*, vol. 48, no. 5, 2015, pp. 26–32.

[30] L. Yao, Q. Z. Sheng, and S. Dustdar, “Web-based Management of the Internet of Things,” *IEEE Internet Computing*, vol. 19, no. 4,2015, pp. 60–67.

[31] S. Bandyopadhyay, M. Sengupta, S. Maiti, and S. Dutta, “Role of Middleware for Internet of Things: A Study,” International Journal of Computer Science & Engineering Survey (IJCSES), vol.2, no.3, 2011.

[32] M. A. Chaqfeh and N. Mohamed, “Challenges in middleware solutions for the internet of things,” *Proceedings of the International Conference on Collaboration Technologies and Systems (CTS),* 2012, pp. 21–26

[33] T. Teixeira, S. Hachem, V. Issarny, and N. Georgantas, “Service oriented middleware for the internet of things: a perspective,” *Proceedings of the Springer Conference on Towards a Service-Based Internet.* 2011, pp. 220-229

[34] M. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Clarke, “Middleware for internet of things: a survey,” *IEEE Internet of Things Journal*, vol. 3, no. 1, 2016, pp. 70–95.

[35] Computing at Edge, http://eecatalog.com/IoT/2015/08/18/computing-at-the-edge/. Last accessed on Jan. 6th, 2018.

[36] A. B. García Hernando, A. Da Silva Fariña, L. Bellido Triana, F. J. Ruiz Piñar and D. Fernández Cambronero, "Virtualization of residential IoT functionality by using NFV and SDN," *Proceedings of the IEEE International Conference on Consumer Electronics (ICCE),* 2017, pp. 86-87.

[37] M. Ojo, D. Adami and S. Giordano, "A SDN-IoT Architecture with NFV Implementation," *Proceedings of the IEEE Globecom Workshops (GC Workshops)*, 2016, pp. 1-6.

[38] T. Wood, K. K. Ramakrishnan, J. Hwang, G. Liu, and W. Zhang, “Toward a software-based network: integrating software defined networking and network function virtualization,” *IEEE Network*, vol. 29, no. 3, 2015, pp. 36–41.

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

[40] O. Flauzac, C. Gonzalez and F. Nolot, "Original secure architecture for IoT based on SDN," *Proceedings of the International Conference on Protocol Engineering (ICPE) and International Conference on New Technologies of Distributed Systems (NTDS),* 2015, pp. 1-6.

[41] K. S. Sahoo, B. Sahoo and A. Panda, "A secured SDN framework for IoT," *Proceedings of the International Conference on Man and Machine Interfacing (MAMI),* 2015, pp. 1-4.

[42] S. Bera, S. Misra and A. V. Vasilakos, "Software-Defined Networking for Internet of Things: A Survey," *IEEE Internet of Things Journal*, vol. 4, no. 6, 2017, pp. 1994-2008

[43] C. Gonzalez, O. Flauzac, F. Nolot and A. Jara, "A Novel Distributed SDN-Secured Architecture for the IoT," *Proceedings of the International Conference on Distributed Computing in Sensor Systems (DCOSS),* 2016, pp. 244-249.

[44] F. Salvestrini, G. Carrozzo and N. Ciulli, "Towards a Distributed SDN Control: Inter-Platform Signaling among Flow Processing Platforms," *Proceedings of the IEEE SDN for Future Networks and Services (SDN4FNS)*, 2013, pp. 1-7.

[45] AMQP, http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-overview-v1.0-os.html. Last accessed on Jan. 6th, 2018.

[46] Constrained Application Protocol [COAP], https://tools.ietf.org/html/rfc7252. Last accessed on Jan. 6th, 2018.

[47] Message Queuing Telemetry Transport Protocol [MQTT], http://mqtt.org/2014/11/mqtt-v3-1-1-now-an-oasis-standard. last accessed on Jan. 6th, 2018.

[48] S. Bandyopadhyay, M. Sengupta, S. Maiti, and S. Dutta, “Role of Middleware for Internet of Things,” *Proceedings of the International Journal of Computer Science & Engineering Survey (IJCSES),* vol. 2, no. 3, 2011, pp. 94–105.

[49] OpenDaylight (ODL) [45], https://www.opendaylight.org/. Last accessed on Jan. 6th, 2018.

[50] IOTDM, https://wiki.opendaylight.org/view/IoTDM_Overview. Last accessed on Jan. 6th, 2018.

[51] Amqp 1.4.9 Python library, https://pypi.python.org/pypi/amqp/1.4.9. Last accessed on Jan. 6th, 2018.

[52] Eclipse Paho MQTT Python client library, paho-mqtt 1.3.1 (latest version support python 2.7 and 3.x both), https://pypi.python.org/pypi/paho-mqtt/1.3.1. Last accessed on Jan. 6th, 2018.

[53] COAP python library openwsn-coap 0.0.2, https://pypi.python.org/pypi/openwsn-coap/0.0.2. Last accessed on Jan. 6th, 2018.

[54] HTTP protocol, python library urllib3, https://pypi.python.org/pypi/urllib3. Last accessed on Jan. 6th, 2018.

[55] Architecture diagram of traditional Gateways ,http://internetofthingsagenda.techtarget.com/definition/gateway. Last accessed on Jan. 6th, 2018.

[56] Freeboard, https://freeboard.io/board/tthn73. Last accessed on Jan. 4th, 2018.

[57] JY901, https://www.aliexpress.com/item/JY901-MPU6050-module-angle-output-9-axis-Accelerometer-Gyroscope-Serial-port-TTL-IIC-Four-rotorpowerful/32605705939. html. Last accessed on Jan. 6th, 2018.

[58] Raspberry Pi3, https://www.raspberrypi.org/products/raspberry-pi-3-model-b/. Last accessed on Jan. 6th, 2018.

[59] Zodiac-fx, https://northboundnetworks.com/products/zodiac-fx. Last accessed on Jan. 6th, 2018.

[60] Sysbench, https://github.com/akopytov/sysbench. Last accessed on Jan. 6th, 2018.

[61] Mbw, https://github.com/raas/mbw. Last accessed on Jan. 6th, 2018.

[62] Bonnie++, http://www.coker.com.au/bonnie++. Last accessed on Jan. 6th, 2018.

[63] PowerTop , https://github.com/XVilka/powertop. Last accessed on Jan. 6th, 2018.

[64] F. Kaup, P. Gottschling and D. Hausheer, "PowerPi: Measuring and modeling the power consumption of the Raspberry Pi," *Proceedings of the 39th Annual IEEE Conference on Local Computer Networks*, 2014, pp. 236-243.

[65] Y. Gao, H. Wang and X. Huang, "Applying Docker Swarm Cluster into Software Defined Internet of Things," *Proceedings of the 8th International Conference on Information Technology in Medicine and Education (ITME)*, 2016, pp. 445-449.

[66] MPSAT, https://github.com/Appdynamics/mpstat-monitoring-extension. Last accessed on Jan. 4th, 2018.

[67] N. Naik, "Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP," *Proceedings of the IEEE International Systems Engineering Symposium (ISSE)*, 2017, pp. 1-7.

[68] A. Foster, “Messaging technologies for the industrial internet and the internet of things whitepaper,” http://www.prismtech.com/sites/default/files/documents/ Messaging-Whitepaper-051217.pdf . Last accessed on Jan. 4th, 2018.

[69] S. Bandyopadhyay and A. Bhattacharyya, “Lightweight internet protocols for web enablement of sensors using constrained gateway devices,” in Computing,

Networking and Communications (ICNC)," *Proceedings of the International Conference on. IEEE*, 2013, pp. 334–340.

[70] T. Jaffey. MQTT and CoAP, IoT protocols, https://eclipse.org/community/ eclipsenewsletter/2014/february/article2.php. Last accessedon Jan. 6th, 2018.

[71] D. Thangavel, X. Ma, A. Valera, H.-X. Tan, and C. K.-Y. Tan, "Performance evaluation of MQTT and CoAP via a common middleware," in Intelligent Sensors, Sensor Networks and Information Processing, *Proceedings of the IEEE 9th International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, 2014, pp. 1–6.

[72] I.Grigorik, "Making the web faster with HTTP 2.0," *Communications of the ACM*, vol. 56, no. 12, 2013, pp. 42–49.

[73] A. Ludovici, P. Moreno, and A. Calveras, "TinyCoAP: a novel constrained application protocol (CoAP) implementation for embedding RESTful web services in wireless sensor networks based on tinyos," *Journal of Sensor and Actuator Networks*, vol. 2, no. 2, 2013, pp. 288–315.

[74] N. S. Han, "Semantic service provisioning for 6LoWPAN: powering internet of things applications on web," Ph.D. dissertation, in Institute National des T´el´ecommunications, 2015.