DISSERTATIONES
INFORMATICAE
UNIVERSITATIS
TARTUENSIS
7

MOHAN LIYANAGE

A Framework for Mobile Web of Things

DISSERTATIONES INFORMATICAE UNIVERSITATIS TARTUENSIS

7

DISSERTATIONES INFORMATICAE UNIVERSITATIS TARTUENSIS
7

MOHAN LIYANAGE

A Framework for Mobile Web of Things

UNIVERSITY OF TARTU
Press

Institute of Computer Science, Faculty of Science and Technology, University of Tartu, Estonia.

Dissertation has been accepted for the commencement of the degree of Doctor of Philosophy (PhD) in informatics on April 4, 2019 by the Council of the Institute of Computer Science, University of Tartu.

Supervisors

Prof. Dr.	Satish Narayana Srirama,
Dr.	Chii Chang,
	University of Tartu,
	Tartu, Estonia.

Opponents

Prof. Dr.	Jussi Kangasharju,
	Department of Computer Science,
	P.O. Box 68, FI-00014,
	University of Helsinki, Finland.

The public defense will take place on June 7, 2019 at 15.00 in J.Liivi 2-405.

The publication of this dissertation was financed by the Institute of Computer Science, University of Tartu.

ISSN 2613-5906
ISBN 978-9949-03-015-6 (print)
ISBN 978-9949-03-016-3 (PDF)

University of Tartu Press
`http://www.tyk.ee/`

To my family and friends

ABSTRACT

The Internet of Things (IoT) represents an environment comprised of various physical and logical objects—such as vehicles, home appliances, animals, manufacturing machines—that have unique digital identifications which enable them to interact with one another over the Internet. Fundamentally, conceptual IoT architecture indicates the Internet Protocol (IP) to be the primary network communication method regardless of what upper layer application protocols are, which leaves the interoperability issues unanswered. In order to address such issues, World Wide Web (W3C) consortium has introduced Web of Things (WoT) that standardises the communication interfaces among IoT entities towards achieving a global standard communication protocol based on the Web technologies.

Meanwhile, mobile devices such as smartphones, tablets, phablets have become one of the primary elements of Mobile IoT (MIoT) systems. Specifically, emerged smartphones have various inbuilt hardware and software sensors, and they are capable of employing as gateways to connect external sensors (e.g., body sensors) to the central management system in the cloud. Therefore, MIoT systems have utilised mobile devices as either sensory data source or as the gateway servers via the Mobile Web Services (MWS) hosted on the mobile devices. MWS enhances the capability of various mobile sensing applications, such as mobile crowdsensing, real-time mobile health monitoring and mobile social network in proximity.

Accordingly, researchers have considered such smart mobile devices as one type of MIoT devices that is capable of connecting and delegating the physical moving objects such as animals, Unmanned Aerial Vehicles (UAVs), land vehicles or even certain levels of human behaviours. Given that there exists various types of mobile devices and application frameworks, the problem of interpretability among MIoT is also raised. To provide seamless communication between MIoT systems, the WoT architecture can be integrated towards adapting MIoT devices to mobile WoT (MWoT) devices.

Although recent mobile devices are quite capable in terms of mobile data transmission speed and computation power, the frequent usage of high-performance multi-core mobile central processing units and high-speed 3G/4G cellular Internet data transmission will quickly drain the battery power of the mobile devices. Numerous existing approaches have sought to overcome the resource-intensive issues in the mobile-embedded service provisioning domain. However, there still exist some challenges that need to be addressed to increase mobile devices' efficiency in terms of energy conservation. To overcome the intensive energy consumption issues of the MWoT, in this thesis, the author presents a lightweight service-oriented framework for MWoT based on constraint application protocols.

Commonly, IoT systems process the data and make the decision at the central management system in the cloud. Similarly, MWoT would follow the same approach. However, considering the latency issue derives from the mobile Internet

speed and the end-to-end latency between the MWoT device and the cloud, the classic model is unable to achieve the applications that demand ultra-low latency in terms of sensory data processing and decision making. In order to support such a need, the author extends the lightweight MWoT framework with mist computing mechanism, which allows the system distributes certain tasks from the cloud to MWoT host. Further, considering the MWoT host have limited hardware components, if it receives multiple requests that involve the same hardware functions at the same period of time, it will cause the conflicts issue and cause the MWoT host unable to satisfy the quality of service. In order to address the issue, the author introduces the resource-aware autonomous service configuration that can manage the availability of the functions provided by the MWoT host, based on the dynamically changing hardware resource availability. Additionally, the framework supports task distribution among a group of mist computing-enabled resources.

In addition, the author has investigated to reduce the energy consumption of MWoT devices, mainly using mobile Internet-based data transmission that consumes more energy than proximal communication methods such as Wi-Fi, Bluetooth Low Energy, etc. Although MWoT device can utilises proximal public Wi-Fi Internet access points, that may raise issues of data security. If MWoT devices can collaborate with proximal Wi-Fi Internet access point that belongs to the pre-identified trusted organisations, they can save significant battery power and mitigate the security issues. Recent research in IoT has emphasised the importance of distributing processes from the distant central server to the proximal resources such as Fog Computing and Networking architecture (i.e., the fog), which can cater the alternative Internet connect for MWoT. However, utilising Fog Networking service (FogNets) is also raises a new research question due to many heterogeneous FogNets providers, that have different capabilities such as operation schedules, workload, and queue size etc. If the MWoT device randomly selects a FogNets provider simply based on the FogNet providers' availability, it may consume unnecessary energy for the MWoT device if the MWoT device switches the connected FogNets provider too frequent. In order to address the challenges discussed above, the author has proposed a proactive FogNets scheduling framework for MWoT. Specifically, the proposed framework aims to optimise the FogNets connection schedule towards reducing the extra energy consumption derived from the switching FogNets providers.

Finally, this thesis extensively addresses the energy conservation of the MWoT. Moreover, we have implemented the proposed frameworks on real devices and evaluated them based on several case studies.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

6LoWPANs	IPv6 over Low Power Wireless Personal Area Networks
BLE	Bluetooth Low Energy
CoAP	Constrained Application Protocol
CoRE	Constrained RESTful Environments
DNS	Domain Name System
EXI	Efficient XML Interchange
FogNets	Fog Networking service
HTTP	Hypertext Transfer Protocol
IoT	Internet of Things
IP	Internet Protocol
JSON	JavaScript Object Notation
LoWPANs	Low-power personal area networks
MEC	Multi-access Edge Computing
MIoT	Mobile Internet of Things
Mist	Mist Computing
MQTT	Message Queue Telemetry Transport
MWoT	Mobile Web of Things
MWS	Mobile Web Services
P2P	Peer-to-Peer
QoS	Quality of Service
REST	Representational State Transfer
RFID	Radio-Frequency Identification
SDM	Service Description Metadata
SOAP	Simple Object Access Protocol
SWoT	Semantic Web of Things
TCP	Transmission Control Protocol
TD	Thing Description
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VM	Virtual Machine
WLAN	Wireless Local Area Network
WoT	Web of Things
WSDL	Web Service Definition Language
WSN	Wireless sensor network
XML	Extensible Markup Language
XMPP	Extensible Messaging and Presence Protocol

1. INTRODUCTION

The Internet of Things (IoT) was first introduced by Kevin Ashton in a presentation at Procter & Gamble in 1999 [111]. According to Ashton's definition, the IoT is a technology that *'empowers computers with their own means of gathering information, observes, identify and understand without the limitations of human-entered data'*. Later, in 2005, the International Telecommunication Union defined the IoT as *'A global infrastructure for the information society enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving, interoperable information and communication technologies'* [70]. Similarly, the European Research Cluster on the Internet of Things defined the IoT as follows: *'The Internet of Things allows people and things to be connected Anytime, Anyplace, with Anything and Anyone, ideally using Any path/network and Any service'* [154].

In contemporary society, the IoT represents an environment that comprises various physical and logical objects—such as computers, mobile telephones, cars, appliances, animals and virtual sensors—that have a unique digital identifier that enables them to interact with each other over the Internet, without requiring human interaction [11]. The ultimate purpose of such an environment is to strengthen and enhance lives and increase the efficiency of various interconnected services and applications in mainly, but not limited to, the three domains of industry, environment and society [11].

For example, supply chain management, transportation and logistics, and aviation can be considered applications in the industrial domain. Some typical cases in this domain are the passive radio frequency identification device (RFID) tag-enabled cards/tags in transportation [74] and supply chain management applications [76]. Agriculture and environmental monitoring can be considered applications in the environmental domain. Common applications in this domain include deploying wireless sensors in farmlands to monitor humidity, temperature, soil moisture, solar radiation and so forth to improve environmental and agricultural sustainability [86], and environmental monitoring systems [87]. Finally, examples in the social domain include applications for healthcare; smart buildings, homes and offices; media and entertainment; and education. A widespread application in this domain is a smart home [65], which enables its tenants to automatically control lighting, cooling/heating systems, windows and blinds, home appliances and so on, which are connected to the Internet.

In contrast to the traditional method of wireless sensor networks (WSNs) that acquire information from the environment, in the IoT, it is expected that things can also interact/react with the physical world [82]. To achieve this requirement, the things may have a type of embedded smartness to function accordingly. For example, smart pill bottles remind patients to take their medication via smart reminders, while records indicate whether the patient has taken the correct dosage with automatic dose tracking[1]. Industry predictions reveal that, over time, the IoT will expand sufficiently to enable thousands of homes and industrial appliances to be connected to the Internet. It is predicted that, by 2020, around 50 billion physical things will be connected to the Internet [20]. Accordingly, the traditional Internet architecture needs to be upgraded to manage the issues encountered when interconnecting billions or trillions of different objects.

Although the IoT continues connecting objects and frameworks to the Internet Protocol (IP)–based Internet using the most current communication technologies—such as Bluetooth, ZigBee, Ethernet and Wi-Fi—there remain some limitations when connecting each and every object. The limiting factor is the non-appearance of a typical technique or interface to establish communication among the different things. To conquer the absence of interoperability across platforms in the IoT, there should be platform-independent application programming interfaces (APIs) that empower unique platforms to know how to communicate with each other. The Web of Things (WoT) is a high-level application protocol designed to maximise interoperability within the IoT. The WoT can be considered a refinement of the IoT to integrate smart things into the Internet and web architecture [63]. According to Tim Berners-Lee, the influence of the web did not reach its full potential until the semantic web was developed. He claimed that the semantic web aims to provide a framework for data and information that can be processed automatically [16].

In the meantime, the W3C WoT [77] is proposed to empower interoperability crosswise over IoT platforms and application areas. It generally provides procedures to formally describe IoT interfaces to allow IoT gadgets and services to speak with each other, separate from their basic implementation, and across multiple networking protocols. Accordingly, WoT systems are essentially IoT systems with mechanisms to empower associated connected things, such as home appliances, vehicles and animals, to be communicable by means of web technologies. In addition, this is considered the familiar application layer of the IoT. Further, the WoT will provide a uniform resource locator (URL) to every thing, and a RESTful API that empowers a gathering of machine-interpretable descriptions of things that enable more physical items to interface with the Internet.

The W3C thing description (TD) [122] approach provides an entry point to a thing that includes rich semantic metadata to describe the thing itself. Moreover, the TD provides a machine-understandable data model and features of the WoT's properties—such as actions, events and interaction models exposed to applications—that provide metadata for communication among the different things over the web. The work of characterising a TD for a current device enables it to participate in the WoT without making any alterations to itself.

In contemporary lifestyles, recently introduced mobile devices—such as smartphones and tablets—have become essential and indispensable tools in everyday

[1]https://nanthealth.com/vitality/

life. With their faster processing power, larger and higher-resolution display screens, greater memory and enhanced power-saving mechanisms, they contribute significantly towards making it easy for us to exist and perform our everyday work. Additionally, accessing high-speed mobile Internet (e.g., 3G/4G or long-term evolution (LTE) Internet) has led mobile device use to become a common phenomenon, with recent reports indicating that about 56% of global web traffic originates from mobile devices [144].

The evolution of service-oriented architecture technologies and enhanced features of mobile telephone–embedded sensors (proximity, light, accelerometer, gyroscopic sensors, etc.), higher computation power and greater storage and actuation capabilities have motivated the mobile sensing paradigm. This has created a new avenue of mobile sensing services, in which smartphones are capable of providing various environmental context information by either pushing the collected sensory data to remote servers or providing the data directly via embedded mobile web services (MWS) technology (also termed the mobile host) [26]. The basic idea of the mobile host is sensing the environment using inbuilt sensors, and providing the sensing services to other clients as a web server. Distinct to the push-based approach, MWS allow remote clients to perform on-demand requests to the mobile host without uploading all the data they have collected periodically.

In addition, contemporary smartphones provide the ability to inventors to build various mobile sensing applications, such as mobile crowdsensing [100, 155] real-time mobile health monitoring [78, 158], real-time activity tracking [14, 108], location based services [36], ambient assisted living [17] and mobile social networks in proximity applications [32, 124]. In general, smartphones play the role of web service consumers. Changing the position of a web service client to a web service provider is a challenging task because of the constrained nature of the devices themselves. While various domains have integrated the IoT into their systems, MIoT [119] emerged as the dominant theme in the IoT. In general, the MIoT specifically addresses connected moving objects, such as humans, animals, drones, and vehicles, which often rely on battery-powered devices. In connected vehicle domain, applications such as Internet of Vehicles (IoVs) based traffic management solution [43], flexible vehicle-to-grid (V2G) coordination scheme to reduce the energy cost of charging stations [160], identify better locations of road-side units for development of Content Delivery Networks [131], can be considered. Also, in a remote healthcare system, the patient must wear various battery-powered body sensors, and the system uses the patient's smartphone to collect the sensory data from the body sensors, and then provides the data to the remote hospital's system [104].

Commonly, if all the MIoT system uses devices from a single vendor, it may have no issues integrating the devices. However, this approach reduces flexibility in choosing the device vendor. Further, if the vendor retreats from the market, the system cannot easily find substitutions, and, eventually, the system manager must replace all the devices. In contrast, if the system has applied the WoT, in which the system uses only WoT standard-compliant devices, it will increase the sustainability of MIoT deployment. To differentiate from the general MIoT, in this thesis, the author terms the WoT-driven MIoT the ‘mobile Web of Things’ (MWoT). MWoT devices also operate as mobile sensing service gateways for other clients. Unlike mobile host devices, which acquire data from inbuilt sensors, the MWoT can collect data from other sensors in proximity and published over the standard web technologies.

The MWoT eases various IoT applications. For example, on the Internet of Vehicles [75], MWoT-compliant connected vehicles can quickly perform thing-to-thing interaction, in which they can discover one another based on multicast domain name system (mDNS), DNS service discovery (DNS-SD) [10] and WoT interfaces to provide autonomous driving safety enhancement. Further, in the ambient assisted living [98] domain, the MWoT can simplify the communication among mobile users and the surrounding environment. For example, a person with a disability can use the MWoT to avoid crowded areas while he or she is in central city streets. Finally, the MWoT eases the deployment and management of remote healthcare [92], so that the associated wearable equipment and environmental devices can efficiently perform communication towards improving sensory data collection for the patient.

1.1. Problem Statement

Although recent smartphones have sufficient processing power and are capable of high-speed data transmission, the challenge of resource constraints still exists when providing MWoT services. The frequent usage of high-performance multi-core mobile central processing units (CPUs) and high-speed 3G/4G mobile Internet data transmission quickly drains the battery power of the MWoT device. Therefore, during recent years, several lightweight MWS provisioning approaches [8, 15, 71, 99] have been proposed to address the resource-intensive issues in the MWoT. Further, numerous frameworks have been introduced to reduce the resource consumption of the mobile host; however, the efficiency of their approach is still limited. In this thesis, the author investigates the following open issues that need to be resolved to adapt the design of future MWoT frameworks.

1. The constrained nature of available resources on mobile devices
   In general, there are two trends of approaches used in most previous frameworks to address the resource-intensive issues in the MWoT:
   (a) Reducing the complexity of messaging, such as using Representational State Transfer (REST)–based service provisioning, rather than Simple Object Access Protocol (SOAP)
   (b) Using external resources to enhance the overall performance, such as offloading complex computational tasks to the static cloud [30] or mobile ad-hoc cloud [7].

However, when considering energy efficiency, there remain issues regarding these frameworks, mainly because of the underlying protocol stack. For example, most of the past frameworks [80, 96, 106, 140] were based on HyperText Transfer Protocol (HTTP) as their application protocol, which is on the Transmission Control Protocol (TCP). Correspondingly, TCP is a connection-oriented protocol that generally has more overhead on the transport layer, and HTTP messages often have large header size (from 200 bytes to 2 kilobytes), which is not suitable for constrained environments.

   - *Which features and lightweight protocols should be involved in a new lightweight MWS provisioning framework to address the limitations in the existing MWoT approaches?*

2. Avoiding conflicts among the sensing services components and increase the quality of service in MWoT
   When considering MWoT-based sensing provisioning systems, at times the workload of the MWoT device may reduce the efficiency of the device and disturb the intended use of the device—for example, if the device is a mobile telephone, it may disrupt telephone calls, the reading of text messages and so on. In addition, the communications latencies in the MWoT will increase when numerous clients are requesting the same information at the same time. Further, mobile devices have limited sensing components and may be unable to operate concurrently. As a result, there may be conflicts among the sensing components. Conversely, some services may be able to operate at the same time, even though they are using the same sensing components. For example, video- and image-based sensing services both use the camera component and may operate at the same time, depending on the specification of the devices.

   - *In such cases, how does the MWoT device manage the services and provide timely service publishing?*

   Similarly, in the case of a real-time or periodical sensing service operation, the executed service can affect the availability of other services.

   - *How does the MWoT device measure or prioritise the availability of the services?*
   - *How does the MWoT device borrow some computational resources from nearby devices, if its own available resources are limited?*

3. Using collaborative fog networking services to reduce mobile Internet usage in MWoT devices
   In general, the IoT environment involves a large number of MWoT devices deployed in high density. Although large-scale connected MWoT devices provide various possibilities, they still raise the challenge of constrained energy because they operate on battery power and use mobile Internet communication.

Correspondingly, as the literature shows [98], Wi-Fi network communication consumes much less energy than cellular Internet, and sending data below a threshold will consume very similar power. For instance, sending 100 bytes or 500 bytes consumes almost the same power via 3G mobile Internet, which indicates that, when a device is brokering a small amount of data for the other proximal wireless local area network (WLAN)-connected devices, it will not consume much extra energy from the device itself.

In this regard, if MWoT devices can collaborate with proximal Wi-Fi Internet access points, they can save significant battery power while engaging in mobile sensing tasks. In the past, many works have discussed such an Internet-sharing environment [9, 11, 42, 52, 69, 149], and the related commercial services have already existed for many years (`e.g.http://www.fon.com`). Moreover, sharing hardware resources has also been discussed in studies [39,42,155] for different pervasive and mobile application domains. Fortunately, recent research on the IoT has emphasised the importance of distributing processes from the distant central server to the proximal resources, such as programmable network routers, gateways, bridges or their co-located machines. In particular, industry terms—such the paradigm of fog computing and networking architecture (i.e., the fog)—can cater to alternative Internet connections for the MWoT.

Since the industry founded the OpenFog consortium, it has been collaborating with the Institute of Electrical and Electronics Engineers (IEEE) to develop the fog standard [69]. It is foreseeable that, in the near future, local small businesses and individuals will be providing public fog to the general public, which is similar to the indie fog business model [28] and The Things Network[2] . Although using the fog networking service is a promising approach to improve the energy efficiency of the MWoT, it also raises new research questions:

- *How can one select the best FogNets provider in an environment that comprises many heterogeneous FogNets providers that have different workloads, queues and operation schedules?*
- *How can one reduce the unnecessary energy consumed by the MWoT device from switching the connected FogNets provider too frequently because of the availability of the FogNet provider?*

1.2. Research Objectives and Contributions

A conventional approach of the MWoT is working as a mobile gateway device that provides some sensing services to other devices over the standard web technologies. However, because of the dynamic nature of the WSN, uncertain mobility patterns and the energy constraints of mobile devices lead to many challenges in

[2]https://www.thethingsnetwork.org/

developing a sustainable MWoT framework. The research presented in this thesis focuses on developing a generic MWoT framework to overcome the challenges mentioned above—particularly energy conservation and avoiding the conflicts of sensing services of the MWoT device. To summarise, the contributions of this work are as follows:

- A lightweight mobile WoT framework
  The framework is based on integrating a number of lightweight protocols, including Constrained Application Protocol (CoAP) [128], Bluetooth Low Energy (BLE) [130] and Efficient XML Interchange (EXI) [73]. The prototype of the proposed framework has been implemented and tested on a number of mobile devices, including Google/LG Nexus 51, Google/HTC Nexus 92 and Raspberry Pi3. The experimental results focus on performance comparison between the traditional MWS and the proposed lightweight MWoT framework, and indicate the efficiency of the proposed lightweight MWS framework concerning resource consumption and service provisioning performance.
- A mePaaS
  As the technology evolved, today's ARM CPU-powered mobile devices can outperform entry-level virtual machines [27] and are capable of supporting such types of platforms. mePaaS nodes can execute customised computational processes defined by their requesters. mePaaS nodes loan their hardware resources to others based on specific service level agreements. Moreover, the framework consists of a module that addresses the hardware usage relative to the temporal space, especially when handling complex types of services. The framework aims to address the two issues described before:
  - a flexible program execution environment in the mobile device
  - self-adaptive resource management; a prototype has been developed based on mePaaS and has been thoroughly tested as a proof-of-concept.
- A proactive FogNets scheduling framework for the MWoT
  In general, in the fog networking [19] environment, devices interconnect with already existing proximal FogNets and use their resources. However, FogNets providers are not always actively connectable as a result of some technical factors, such as maintenance and energy preservation. The proposed framework is mainly designed for such environments. Establishing a collaborative network in such an environment requires an appropriate scheduling scheme. Specifically, the proposed framework aims to optimise the FogNets connection schedule towards reducing the extra energy consumption derived from switching the FogNets providers. The author implemented and validated the proposed framework using physical devices, and the experimental results indicate that the proposed architecture can reduce the energy consumption for MWoT devices.

1.3. Outline

The chapters in this thesis are organised as follows.

Chapter 2 summarises the state-of-the-art and related work in the MWoT. In particular, the first section describes related lightweight operating systems for sensor networks, such as Contiki and TinyOS. The following sections describe some of the related technologies in lightweight communication, WoT architecture and lightweight middleware for the MWoT. The author then introduces related works that enhance the service provisioning of MWoT frameworks—such as using nearby resources in the MWoT to preserve energy, improving energy-efficient approaches in WSNs and using public fog networking services. In the latter part of Chapter 2, the author introduces further mechanisms to preserve the energy of the MWoT by exploring energy-efficient service description and service discovery approaches.

Chapter 3 introduces a lightweight MWS provisioning framework that uses lightweight communication protocols and a scheduling algorithm to enhance the energy efficiency of the MWoT devices. The proposed framework in this chapter was partially published in a previous article ('Lightweight Mobile Web Service Provisioning for the Internet of Things Mediation' [88]).

Chapter 4 describes a service-oriented mePaaS—a framework that enables edge MWoT devices to support resource-aware autonomous service configuration, which can manage the availability of a platform that allows requesters to deploy and execute their own program models. Partial contents of this chapter have been previously published in [90,91].

Chapter 5 discusses energy-efficient mobile data acquisition using public Fog networking services. The framework introduces a proactive FogNets scheduling scheme in the opportunistic Internet-sharing environment. MWoT devices can continually retrieve information of already existing proximal FogNets and schedule the connection among the available FogNets providers accordingly. The proposed architecture in this chapter has been partially published in [89].

To evaluate the proposed frameworks, Chapter 6 presents the details of prototypes that the author has implemented on real devices, and the evaluation results of the use cases.

Chapter 7 concludes the thesis by summarising the contributions and evaluation results, and by outlining the further work that should be undertaken to enhance the MWoT service provisioning.

2. STATE OF THE ART

Providing MWS in which the mobile terminals are used as both web service clients and providers is not a new concept, and has been a popular research area in recent years. This chapter explores the state-of-the-art options available in the MWoT, particularly in relation to system architectures, protocols and communication mechanisms.

2.1. Lightweight Operating Systems for Sensor Networks

In general, WSNs consist of a large number of resource-constrained tiny sensor nodes that collectively form a network to acquire and transport sensor data. For a large number of node deployments, it is not practically feasible to frequently move to every node and change/update their configuration when necessary. To address these types of issues, researchers have proposed using lightweight, tiny operating systems for sensor nodes. The following subsections describe two of these commonly used operating systems.

2.1.1. Contiki

Contiki [49] is a lightweight operating system developed for typically constrained sensor nodes, with the feature of loading and unloading individual programs and services dynamically. The operating system is implemented in the C language with event-driven kernel architecture, and can be ported to conventional sensor devices with 8-bit microcontrollers, less than 20 kilobytes of random-access memory (RAM) and a few hundred kilobytes of read-only memory (ROM). For instance, Contiki was simply ported to the ScatterWeb [121] embedded sensor board with the configuration of the Texas Instruments MSP430 series, an ultra-low-power microcontroller, 2 kilobytes of RAM and 60 kilobytes of ROM running at 1 MHz. Moreover, instead of downloading a complete binary image of the entire system, Contiki is built with the ability to load and unload distinct applications or services at runtime over the network. Such a feature is essential to reduce the number of bytes delivered over the network, which consumes less node energy and requires less transfer time.

2.1.2. TinyOS

TinyOS [67] is a tiny micro-threading operating system that provides lightweight thread architecture and efficient network interfaces. The main design goals of the TinyOS are offering a flexible, application-specific operating system that provides efficient hardware and application modularity and maintains simultaneous different data flows. Moreover, it tackles the challenges of the limited memory and power resources of sensor networks, while providing a flexible, fine-grain execution model that supports concurrent operations with insufficient memory and power constraints.

2.2. Lightweight Communication Protocols

Energy efficiency in the WSNs—specifically in the MWoT—is an active research domain, and lightweight communication protocols are playing a significant role in the energy preservation of the whole system. The following subsections discuss a few communication protocols involved in the MWoT.

2.2.1. Micro IP (uIP)

The uIP [48] is a small, generic and portable TCP/IP implementation, with the absolute minimal set of features required for full TCP/IP stack. The application is written in the C programming language, and supported only for a single network interface. Moreover, uIP implementation supports devices running on various 8/16-bit platforms, with the features for handling generic TCP/IP protocols stacked on devices with limited resources. Further, it is possible to run the uIP on devices with as little as 200 bytes of RAM, with limited performance. However, the initial implementation of the uIP focuses on IPs, Internet Control Message Protocols and TCPs, and does not support User Datagram Protocols (UDPs).

2.2.2. 6LoWPAN

For applications that require low-cost communications and their nodes to work autonomously for years on small batteries, the IEEE 802.15.4 introduced a wireless link for low-power personal area networks (LoWPANs). Further, the IEEE 802.15.4 designed the link explicitly for low-cost, long-lived application domains, and includes features such as low transmit power, small frame sizes, low bandwidth, limited memory and limited computing power [68]. Specifically, the LoWPAN's frame length is limited to 128 bytes, 16-bit link addresses and low throughput (250 kbps in the 2.4 GHz band and 20 or 40 kbps in other bands), which ensures reasonably low header overhead and memory requirements. The microcontrollers that associated with the LoWPANs implementations and enough to have about 8 KBytes of data RAM and 64 KBytes program ROM. However, integrating IPv6 over LoWPAN networks raises several challenges, mainly because of the bulky nature of the protocol headers and because some characteristics of the IPV6 do not fit in the low-power network. To overcome this issue, one can principally reduce the header size when transmitting IPv6 datagrams over the 802.15.4 links. As shown in Figure 2.1, the Internet Engineering Task Force (IETF) introduced the 6LoWPAN [83], which consists of an adaptation layer between the link and network layer of the LoWPAN protocol stack. The 6LoWPAN achieves low overhead by providing mechanisms for the header compression and fragmentation of the IPv6 packets into multiple link-level frames.

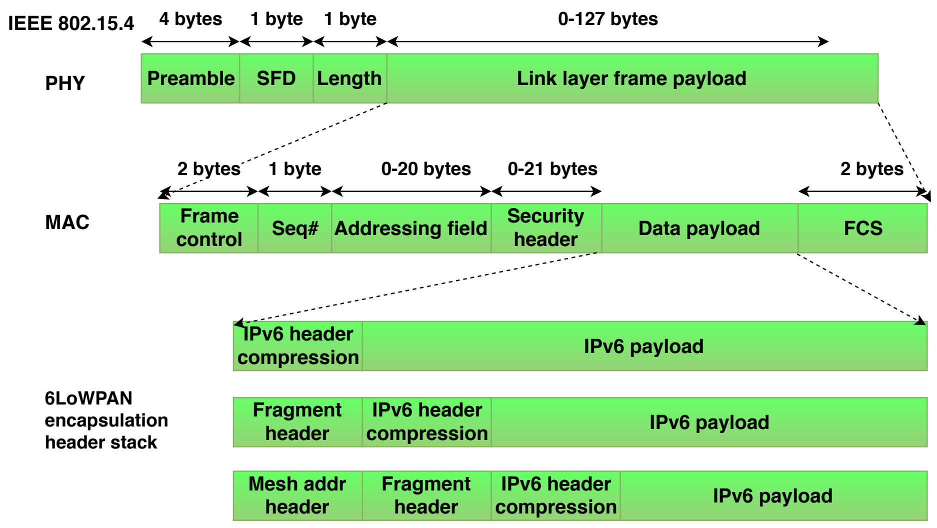


Figure 1: 6LoWPAN encapsulation header stack

2.2.3. Representational State Transfer (REST)

REST [54] is a communication protocol architectural style for distributed hypermedia systems that handles client–server communication over the web. A web service designed based on REST is termed a 'RESTful' service. RESTful services are mainly concerned with the system's resources transferred over the web. Uniform Resource Identifiers (URIs) are used to access the resources on the web server, and provide a simple mechanism for communication between applications over the Internet. For example, a sensor node with a sensor to measure the ambient temperature and a light-emitting diode (LED) to trigger an alarm can have the following URIs:

`/office-room-1/sensor-node1/sensors/temp` (represents the temperature sensor)
`/office-room-1/sensor-node1/actuators/led1` (represents the alarm LED)

In contrast to the more complex mechanisms—such as the Common Object Request Broker Architecture (CORBA) [50], Web Services Description Language (WSDL) and SOAP—the RESTful application uses the following basic HTTP methods to access web resources:

- `POST` - to create a new resource
- `PUT` - to update the existing value/state of the resource
- `GET` - to access the current value/state of the resource
- `DELETE` - to remove the resource

With the HTTP methods and URIs mentioned above, developers can design systems that can easily interact with resources over the web. For instance, a web-based smart office system that can measure room temperature at regular intervals

and trigger an alarm if the temperature exceeds the threshold (e.g.,35^0C) can operate on the following URIs with the relevant HTTP methods:
A `GET` on the URI
`http://abc.com:8585/office-room-1/sensor-node1/sensors/temp`
reads the current value of the temperature sensor.
A `PUT` on the URI
`http://abc.com:8585/office-room-1/sensor-node1/actuators/LED1` with the value of `{status=1}` can turn on the alarm LED.

Moreover, the experimental results in [8, 140, 151], indicated that REST is more suitable for resource-constrained MWoT environments. For instance, in the work [8], the authors have compared SOAP-based and RESTful-based mobile Web service provisioning frameworks and confirmed that RESTful-based mobile Web service framework is more suitable for providing mobile Web Services. Moreover, the evaluation results have shown that RESTful Web services consume lower processing time, a higher threshold value for concurrent client requests and less amount of consumed memory compared to SOAP-based Web services. In addition, SOAP requests require heavyweight parsers that consume more processing power while RESTful-based Mobile Web servers required lightweight processing power which allows it to provide continuous mobile Web service provisioning in a reliable manner.

2.2.4. Constrained Application Protocol (CoAP)

CoAP is a RESTful web transfer protocol that is designed for constrained environments. CoAP provides a more compact message format mechanism, which introduces a low overhead binary encoded header and reduces the message parsing complexity. Figure 2 illustrates the underlying protocol stack of CoAP. The significant features of CoAP are:

- a request/response (client/server) communication model
- unicast and multicast support
- a four-byte header
- support of the asynchronous transaction model
- inbuilt service discovery
- four REST request methods—GET, POST, PUT and DELETE
- a subset of Multipurpose Internet Mail Extensions (MIME) types and HTTP response codes
- four different message types—confirmable, non-confirmable, acknowledgment and reset
- URI-based resource representations for easy resource discovery
- built on top of a UDP that has a significantly lower overhead with multicast support in contrast to the TCP; recently, a work-in-progress CoAP-over-TCP solution has been introduced [115].

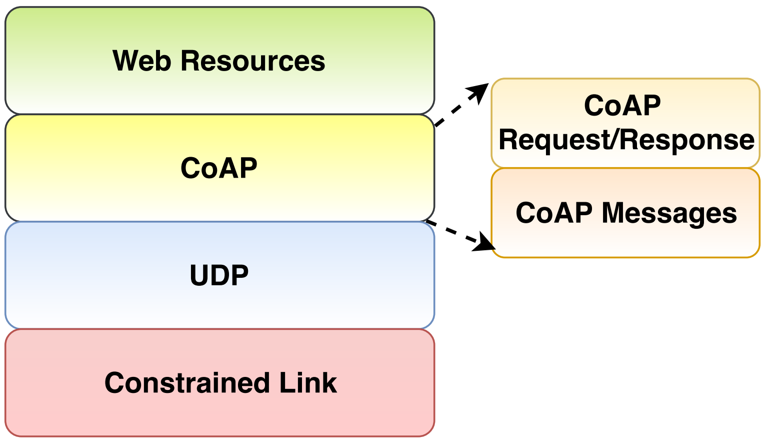


Figure 2: CoAP protocol stack

CoAP is organised in two layers:

1. The transaction layer, which handles message exchange between endpoints. Message exchange on the transaction layer has four types:
    a) confirmable (must be confirmed with an acknowledgment [ACK] or reset [RST] message)
    b) non-confirmable (unreliable exchange, does not need to be acknowledged)
    c) acknowledgment (used to acknowledge a confirmable message)
    d) reset (a confirmable message when the recipient of a message encounters an error).
2. The request/response layer, which is responsible for the transmission of requests and responses. This layer is mainly based on the REST architecture.

With this dual-layered approach, CoAP provides high reliability, even without using TCP as the transport layer protocol. Further, CoAP uses a stop-and-wait approach to deal with packet losses. For example, a CoAP endpoint repeats a request if it does not receive an ACK (or RST) message when time-out expires and the transmission counter is less than four. Figure 3 indicates a reliable message transmission between the client and the server.

In general, CoAP also uses URIs and REST methods to publish/discover the resources hosted by the servers. For example, a GET on the URI
`coap://myserver.com:5683/office-room-1/sensor-node1/sensors/temp`
can be used to retrieve the current data on the resource (temperature sensor). Further, CoAP endpoints use the Constrained RESTful Environments (CoRE) [126] link format for service discovery, as will be discussed in a later section.

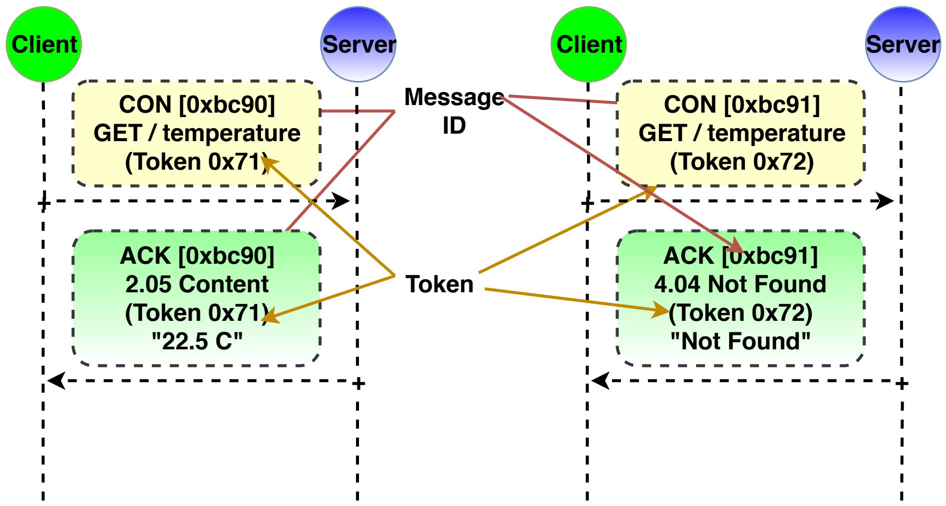


Figure 3: CoAP Request/Response Examples

2.2.5. Message Queue Telemetry Transport (MQTT)

MQTT [13] is also a lightweight message/data-streaming protocol designed for resource-constrained devices, such as sensors and actuators, to send data over low-bandwidth networks. MQTT does not specify a particular data format for the information exchange procedure. It has a typical mechanism of the message-oriented paradigm that uses a topic-based publish/subscribe model, which means that multiple clients can establish connections with a broker node, publish messages to topics, and subscribe to specific topics of interest. Moreover, the exciting feature of MQTT is that the same MQTT client can play a dual role as a publisher and subscriber. The main features of MQTT are as follows:

- lightweight message-queuing protocol with two-byte header
- operates in connection-oriented transport approach (TCP)
- has an asynchronous communication model
- has a publish/subscribe model that decouples the data producer from the data consumer.

When considering the publish/subscribe model of MQTT, sensors mainly publish data on specific topics to a broker, and the clients who subscribe to the particular topic receive the sensor data from the broker. One of the significant advantages of this architecture is the decoupling of the clients needing data and the sensors sending data, which indicates that the sensor nodes do not need to know the clients that are interested in their data, and, conversely, clients need not know the identities of the sensor nodes that are generating the sensor data. For instance, multiple applications may need to retrieve data from a particular sensor for different purposes. All the sensor needs to do is publish its data to the broker on the appropriate topic. The broker will then redistribute the published data from the sensor to the applications that have subscribed to the same topic. Figure 4 describes the general publish/subscribe model of the MQTT protocol.

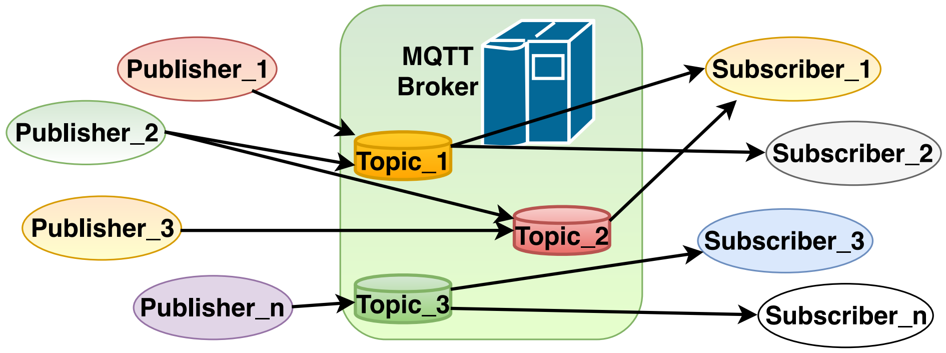


Figure 4: MQTT Pub/Sub communication model

Another advantage is that applications are kept isolated from the changes /failures that occur at the publisher's end. Specifically, in WSNs, this feature is essential because it is prevalent to see device failure and replacement with new nodes. From the application developer's viewpoint, this publish/subscribe model hides the complexity of the underlying network architecture, and the developer only needs to know the topic that the device uses to publish its data. Usually, MQTT does not care about routing or networking strategies, and assumes that the underlying network provides the facilities to exchange messages. MQTT topics are arranged hierarchically, and the publisher publishes the topic to the broker, and the interested subscribers contact the broker to attain the relevant subscription. For example, in a smart building system, a temperature sensor publishes ambient temperature on the topic:

```
building101/floor1/room102/temperature
```

and the clients who are interested in the temperature in Room 102 can connect to the message broker and subscribe to the same topic.

Correspondingly, according to the syntax, MQTT topics must have at least one character to be valid and must be case sensitive. Further, the topic uses forward slashes (/) to separate the topic hierarchy. A few example topics include

```
myhome/floor1/SithmiBedroom/light
myoffice/floor3/room311/temperature
Estonia/Tartu/Ulikooli/ITA/mclab/door1
```

When clients subscribe to the topics, it can be the exact topic where the interesting message was published, or the client can subscribe to more interesting topics at once. For that purpose, MQTT provides two different types of wildcard: single level (+) and multi-level (#). With the single-level wildcard (+), a client can subscribe to a topic and substitute for one topic level only. For example, a client can subscribe to the topic:

```
myhome/floor1/+/temperature
```

and receive information about the temperature in different locations with just a single subscription. Some example topics that can subscribe to the above syntax are as follows:

`myhome/floor1/livingroom/temperature`
`myhome/floor1/bedroom1/temperature`
`myhome/floor1/bedroom2/temperature`

However, the single-level wildcard does not allow the client to receive data published on topics such as:

`myhome/floor1/bedroom1/humidity` or
`myhome/floor1/livingroom/heater1/temperature`

Accordingly, when a subscriber needs to subscribe to more than one topic level at once, it can use a multi-level wildcard, which covers an arbitrary number of topic levels. For example, when a subscriber is subscribed to a certain topic, such as `myhome/floor1/#`, with wildcard #, it will receive all the information published on the topic`myhome/floor1/` down to all the levels. As a result, the subscriber will receive the content from the following topics:

`myhome/floor1/bedroom2/temperature`
`myhome/floor1/livingroom/humidity`
`myhome/floor1/bedroom1/temperature`

However, the subscriber will be unable to receive messages published on the following topics:

`myhome/floor2/bedroom1/temperature`
`office/floor1/room311/humidity`

Additionally, it is recommended to use more specific names for topics, instead of general topic names. Although the MQTT uses TCP and IP as the underlying protocols, compared with HTTP, the header size of the MQTT message is minimal (i.e., two bytes).

Although TCP provides the reliable transport of messages, MQTT also defines three quality of service (QoS) levels (Figure 5) to ensure message delivery:

QoS Level 0—messages are delivered at-most-once (best effort), based on the guarantee provided by the underlying network (TCP/IP). Frequently used within non-critical applications, which are tolerated for the loss of a few data items.

QoS Level 1—provides at-least-once delivery. Messages are guaranteed to be delivered, but possibly with duplicate messages.

QoS Level 2—messages are delivered exactly-once. This level includes more overhead than the other two types because of control messages and the need to store messages locally.

Further, MQTT has a retained messages facility, in which the broker keeps messages even after they are sent to the subscribers. If a new subscription for the same topic occurs, retained messages can be automatically sent to the new subscriber.

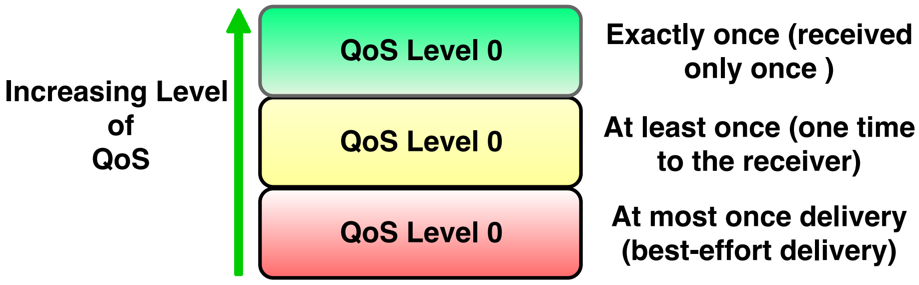

Increasing Level of QoS
QoS Level 0
QoS Level 0
QoS Level 0
Exactly once (received only once)
At least once (one time to the receiver)
At most once delivery (best-effort delivery)

Figure 5: QoS levels in MQTT

2.2.6. Extensible Messaging and Presence Protocol (XMPP)

XMPP is an XML-based streaming protocol that was developed by the Jabber open-source community [117]. It provides facilities for users to communicate with others in real time and attain their status information, such as available, away or offline. XMPP uses the fundamental architecture of the client–server model, whereby a client with a unique name connects to another client with a unique name across a server that provides the routing facility to exchange messages. There may also be servers that can connect to other servers to enable inter-domain communications. For example, when a client in domain abc.com needs to establish communication with a client in domain xyz.com, the XMPP server of domain abc.com routes the message to the XMPP server in xyz.com. Further, XMPP provides translate facilities for different messaging domains and protocols through the XMPP gateways. For example, Figure 6 displays an XMPP network with gateways that translate between a short message service (SMS) domain and a simple mail transfer protocol (SMTP) domain.

According to the XMPP addressing scheme, participants on the XMPP network have a unique XMPP ID called a Jabber ID (JID). The structure of the JID is similar to an email address, with a username and domain name (i.e., username@abc.com). There is a possibility that a single user may log in to multiple locations within the same domain. To identify various log-in locations for the individual user, Jabber specifies a resource for a particular address that denotes a location.

2.3. Web of Things Architecture

The primary focus of the WoT is to overcome the interoperability issues across IoT platforms and applications by providing a web-enabled platform-independent standards framework for IoT devices and services. As a step in this direction, some recent research activities [18,60,62,63,97] discussed platform-independent APIs for application developers that enable different IoT platforms to discover and inter-operate with each other, independent of their underlying protocols.

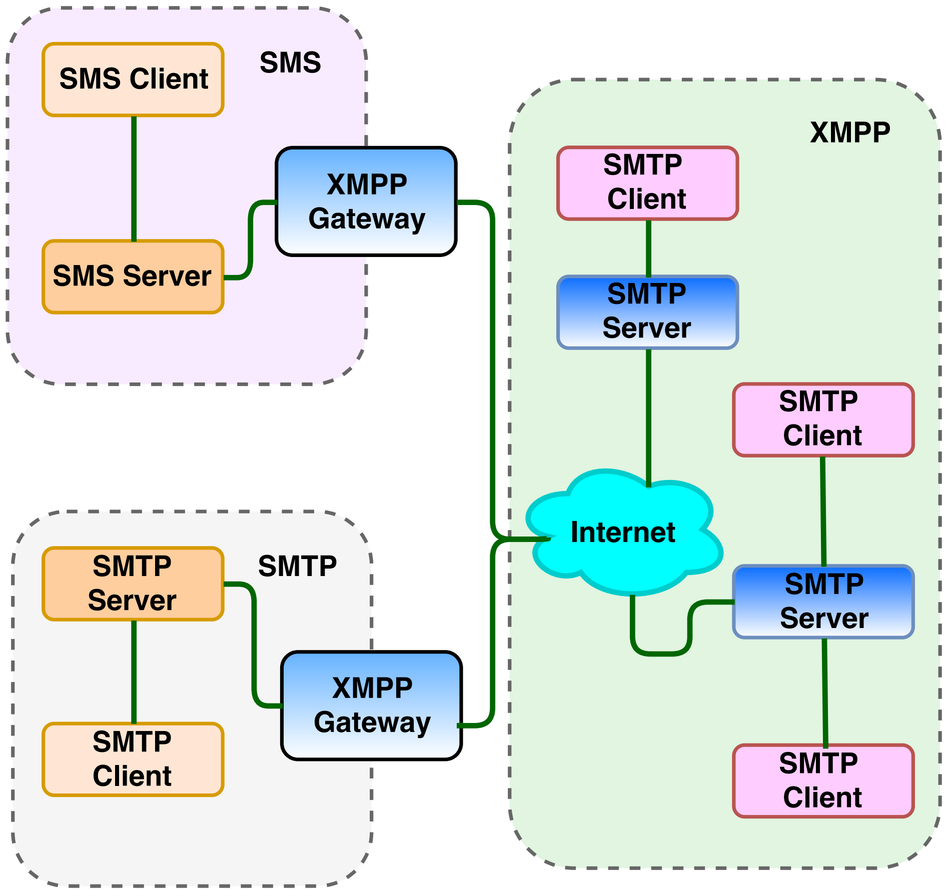


Figure 6: XMPP core architecture

For example, [63] described a WoT framework based on the RESTful architecture, which is already succeeding in web applications. Moreover, the authors proposed embedding a tiny web server on smart things, including the REST principles, where the smart things and their functionality acquire transportable URIs. As a result, the smart things can discover, link together, interact with each other and provide services entirely over the web browser across multiple networking protocols.

To integrate the smart things with existing services on the web, [60] proposed a four-layered WoT architecture based on the device accessibility layer, findability layer, sharing layer and composition layer. Essentially, the layered architecture delivers more flexibility and customisation possibilities for application developers to address interoperability issues. The device accessibility layer explicitly provides a consistent way to access all types of connected objects. The author explored a method of using RESTful architecture to connect incompatible devices to the Internet and push data from smart things to clients.

The primary goal of the findability layer is to tackle issues in searching for relevant services among smart things, and integrate this into applications. However, it becomes unfeasible to make them searchable and available their services to be discoverable when considering large-scale networks, where billions of smart devices are connected. Consequently, the smart things metadata model [61] enables smart things to describe their services on the web so they can be discovered automatically, and both humans and machines can recognise the services they provide. While the device accessibility layer of the WoT architecture provides the perfect integration of smart things, and the findability layer enables them to be discoverable by the world, there remain concerns about public sharing. Allowing access to these services and devices from the web in an open manner has highlighted critical privacy and security breaches.

Finally, the top layer—the composition layer—proposes adopting a Web 2.0 Mashup Editor to the WoT application, and further defines the notion of physical mashups. This layer provides a unified view of the WoT, and expands the boundaries closer to the developers and to end-users for easy integration of applications on top of smart things. Recently, the W3Cs WoT working group presented the first standardised specifications of WoT, which included the WoT TD, WoT binding templates and WoT scripting API [77]. The draft denotes the general architecture and terminology of the W3C WoT building blocks, which can be applied at three levels, as the device level, gateway level (or 'edge') and cloud level.

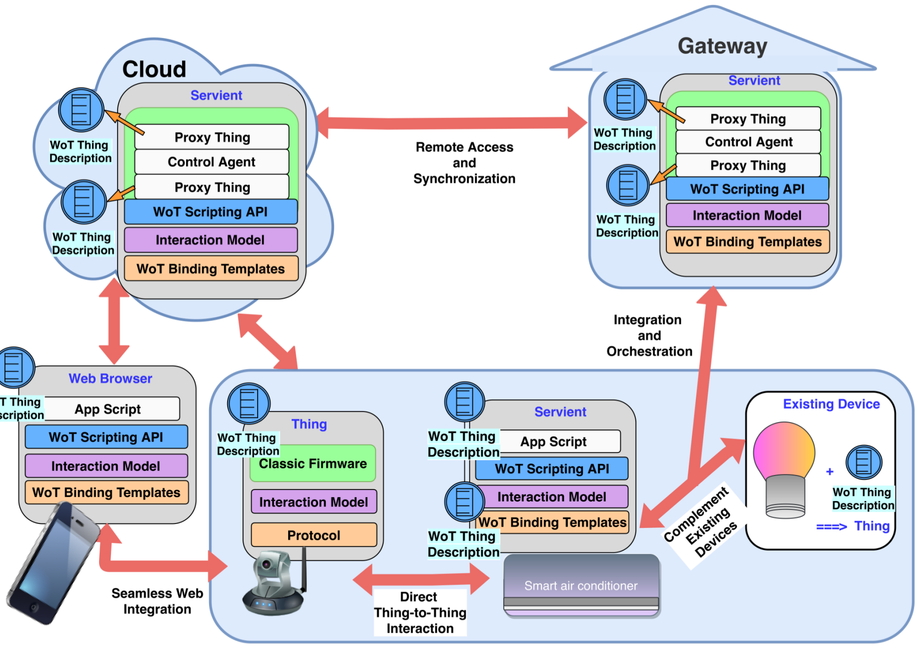


Figure 7: Abstract Architecture of W3C WoT

2.3.1. Web of Things Building Blocks

The WoT building blocks[1] describe the abstract architecture for the WoT. Figure 7 illustrates the three levels of WoT as:

- the device level
- the gateway level (or 'edge')
- the cloud level.

At these levels, the WoT building blocks can be applied with a high-level functionality for each of them at each level. In addition, Figure 2.8 displays the way the WoT building blocks are used in the WoT architecture.

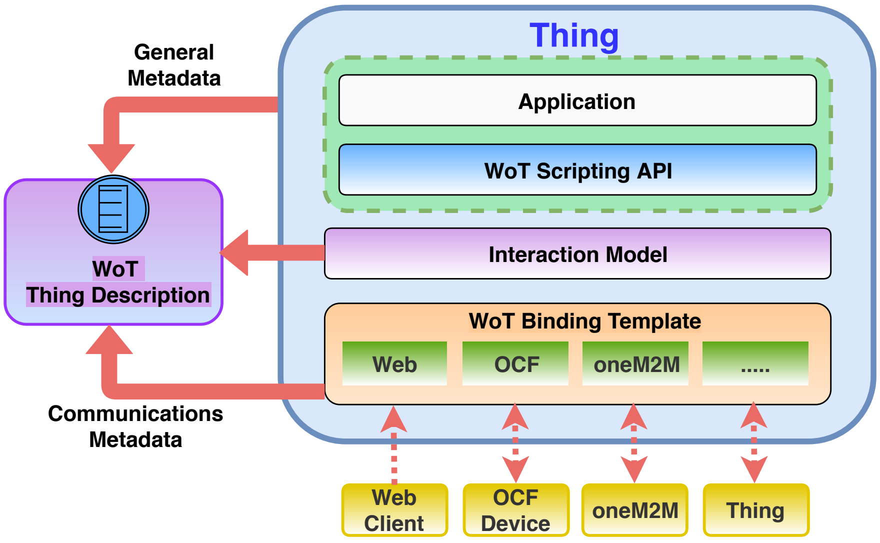


Figure 8: Conceptual Architecture of the WoT Building Blocks

Thing

A thing is the abstraction of a physical or virtual entity, such as a device, a component of hardware or a location embedded in IoT applications. A thing must have a TD and may provide a network-facing API (WoT interface) for the interaction.

Web of Things 'Thing' Description

A TD offers domain-specific metadata of a thing, which further describes the thing's interactions, data model, communication, security mechanisms and so forth. TDs enable developers to use a common, uniform format that provides all the necessary details to access and use IoT devices' data, and empower machine-to-machine communication in the WoT. The WoT TD is developed on the formal interaction model, which consists of the default interaction patterns, such as property, action and event. The properties include the data points of a thing that can read or write, actions represent the executable processes, and events involve the

[1]http://www.w3.org/TR/wot-architecture/#sec-building-blocks

interactions of remote endpoints when pushing data asynchronously. TDs are aligned with the CoRE Resource Directory [127] lookups by using thing directories that provide a web interface for developers to smooth interactions with things. Currently, by default, TDs serialise to JavaScript Object Notation for Linked Data (JSON-LD) [38], which offers machine-understandable semantics for things.

Web of Things Binding Templates

The WoT binding templates provide a means to interact between different IoT platforms by including a rich set of communication metadata in the TD. When defining a TD for a particular device, the corresponding binding template of the IoT platform can be used, and, for each IoT platform, it is sufficient to create a WoT binding template only once and reuse it with all the devices' TDs. Figure 9 displays the way that binding templates are connected with the TDs and then deal with different protocols.

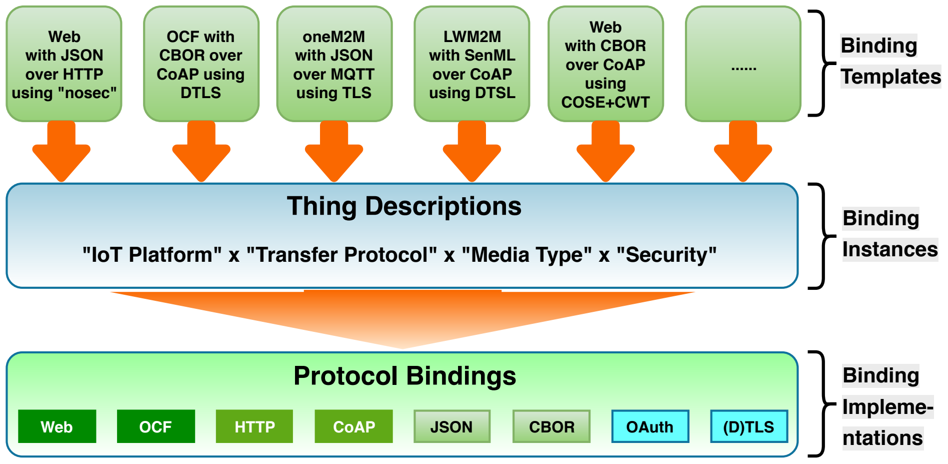

Web with JSON over HTTP using "nosec"
OCF with CBOR over CoAP using DTLS
oneM2M with JSON over MQTT using TLS
LWM2M with SenML over CoAP using DTSL
Web with CBOR over CoAP using COSE+CWT
......
Binding Templates
Thing Descriptions
"IoT Platform" x "Transfer Protocol" x "Media Type" x "Security"
Binding Instances
Protocol Bindings
Web
OCF
HTTP
CoAP
JSON
CBOR
OAuth
(D)TLS
Binding Impleme-ntations

Figure 9: From Binding Templates to Protocol Bindings

Web of Things Scripting Application Programming Interface

The WoT scripting API is a runtime system, like a web browser, built on top of the thing abstraction and the TD interaction model. It enables developers to reduce integration costs and increase productivity. There are three sub-APIs:

- WoT—object work as an entry point of the API that allows discovery, consumption and exposure of things
- ConsumedThing—the client API works as an interface to consume things over the network or locally (e.g., physically attached hardware)
- ExposedThing—works as server API that provides an interface to configure and expose things over the network.

Mainly, the WoT TD describes the network-facing interface of a thing (WoT interface) and works as the primary building block of the WoT implementations. These templates provide multiple possible protocol bindings in which the things can interact with heterogeneous IoT platforms and simplify IoT application development.

2.4. Lightweight Middleware for Mobile Web of Things

Mobile devices, such as smartphones and tablet computers, have now become essential to many people's everyday lives. Over the years, smartphones with multiple features have evolved to become more versatile companion devices. In the same manner, the rise of it in all industries, smartphones continue to play an important role when there is a need to collect data from sensors around humans and interact with them. Mobile devices will eventually become the primary user interface for applications, as they provide real-time data, facilitate an efficient method of communication, process data on the site and so on. For example, smartphones play the role of the 'brain' in the body area network [152] concerning their enhanced storage and communication facilities. However, there are numerous challenges to enhancing smartphones' role because of the constrained nature of the available resources on mobile devices. It is expected that web services hosted on mobile devices must have acceptable performance and have no influence on the primary use of mobile devices, such as making a telephone call, sending text messages and accessing applications. This section mainly discusses the literature regarding MWoT service provisioning frameworks.

2.4.1. Applications of Mobile Web of Things

The MWoT has been used in different application domains in recent years. Berger et al. [15] were successful in provisioning a SOAP-based retail web application, whereby a mobile device presented wallet services to an electronic checkout at the kiosk. The kiosk station used a mobile wallet service hosted by the customer mobile device, which delivered a complete payment transaction within the mobile device, retail store system and bank system. Two other fascinating applications were: (a) a mobile picture album with location information service and (b) cooperative journalism, where the mobile host engaged the coordination of journalists and their associations. These applications were presented in work by Srirama et al. in [138] and [137], respectively. MobiCrop [94] is an application that allows crop farmers to use their mobile devices to gain information about using pesticides. Another interesting mobile health application provided in [96], called SOPHRA, supports healthcare professionals to access patients' medical information that is hosted on their mobile devices.

2.4.2. Comparison of Existing Mobile Web Services Frameworks

Most of the earlier approaches were based on providing a SOAP-based MWS architecture. The work in Srirama et al. [134, 138, 139] presented a lightweight mobile host with Personal Java implementation. In that work, the authors used

HTTP and TCP for the application and transport layer protocols, respectively, and employed a SOAP message format with the WSDL as service description. Gehlen and Pham [56] presented a peer-to-peer (P2P) web service on Java-enabled mobiles. In this architecture, each mobile node operated in both a client/server role, which provided service in the distributed environment. The authors developed a mobile SOAP server capable of providing XML web services. In another work, Srirama [135] and Srirama et al. [140] described a framework based on JXTA technology for service provisioning in the P2P network environment. The light version of JXTA for mobile devices, called JXME, was applied to establish a virtual P2P network. However, JXTA uses either TCP or HTTP protocols to traverse network barriers, and the work also used WSDL for service discovery and SOAP message format. Moreover, Ou et al. [106] designed a layered P2P architecture to provide web services in converged cellular and ad-hoc network environments, with a vertical tunnelling model to speed up the service discovery. The framework consisted of eight modules, including a WSDL module for service publishing and a SOAP module for generating and parsing SOAP messages between the client and service provider.

To reduce the workload at the mobile host, Kim and Lee [80] proposed sharing a task between devices. The authors suggested that migration and replication of web services are necessary to provide a reliable service. The migration or replication of a service is initiated by the service provider, when a request is received and there is a change in its context information. For instance, variations of context information include a shortage of battery level or change in the device location, which stops the device from providing the service on its own. However, although migration of the services reduces the workload at the mobile, the communication is still based on the same traditional protocols, such as HTTP, TCP, WSDL and SOAP, which are heavyweight.

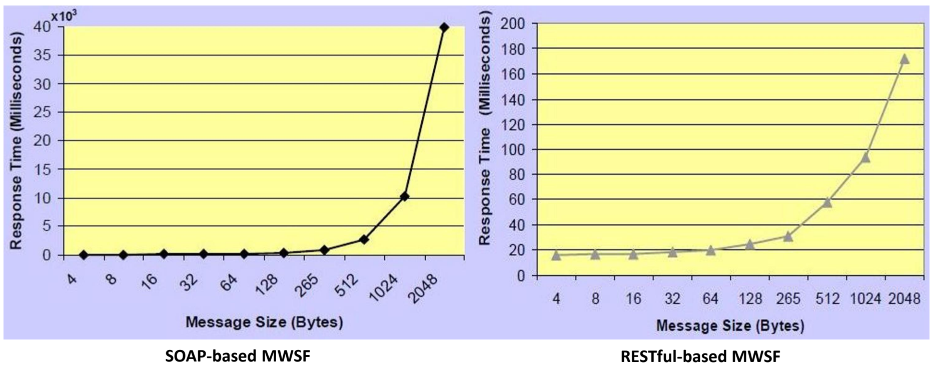


Figure 10: Effect of message size on process time - SOAP Vs. REST [8]

The RESTful web service approach has been used in later mobile host frameworks, and is more appropriate for constrained conditions because of its lightweight features. For instance, AlShahwan and Moessner [8] compared SOAP-based and

RESTful-based mobile Web service provisioning and confirmed that RESTful-based mobile Web service framework is more suitable for the mobile environment (Figure 10). Also, Chang [23] proposed a context-aware cache pre-fetching strategy for MP2P web service provisioning. Instead of XML-formatted documents, such as SOAP, this framework used a RESTful architecture, which is more suitable in the infrastructure-less decentralised MP2P environment. Ali et al. [6] presented a distinctive approach that will reduce the usage of mobile resources and minimize the computation and transmission overhead when offloading services. In addition, the authors have presented that REST-based services are more suitable than SOAP Web Services in terms of execution time and energy consumption (Figure 11).

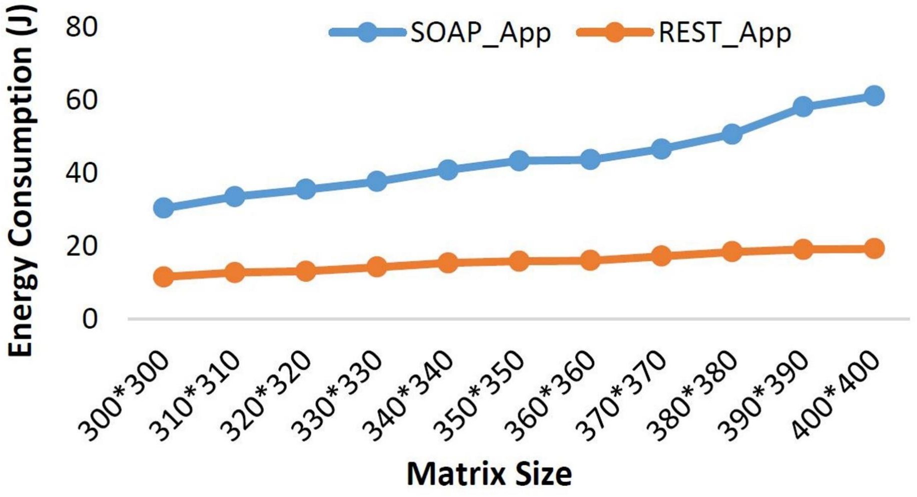


Figure 11: Comparison of energy consumption between SOAP and REST Web Service [6]

In addition, Srirama and Paniagua [142] proposed a mobile host for Android devices based on the Open Services Gateway initiative (OSGi) framework, which is capable of providing services in a RESTful manner using HTTP and XMPP.

Mohamed and Wijesekera [99] proposed a lightweight framework for providing web services on mobile devices, based on the REST architecture, with JSON data representation. Moreover, the framework consisted of main elements such as ‘HTTP Request/Response Listener’, ‘Deployment Agent’ and ‘Request Processor’ to integrate the web services, and ‘Security Agent’ to provide security services for the web services. In addition, Charl van der [151] presented another framework that enabled context-aware services by using a hybrid model of mobile P2P and JmDNS for service discovery. Accordingly, the JSON-formatted messages reduced the communication overhead between devices, and the multicast interchanges messages increased the reliability of the proposed framework.

Lomotey and Deters [95] have studied different flow patterns of sensor data and their implications for energy consumption. There are three main flow patterns were defined as 1) Sequential (the smartphone communicates with the sensors within the edge architecture one-device-at-a-time), 2) Parallelism (request is sent to all devices within the edge but responses are retrieved based on the minimal value of the request-response travel time + processing time), and 3) Choice (connecting to a sensor based on the least request-response travel time and ignoring the processing time component). The preliminary result has shown that the flow pattern with the highest processing demand consumes more power (Figure 12).

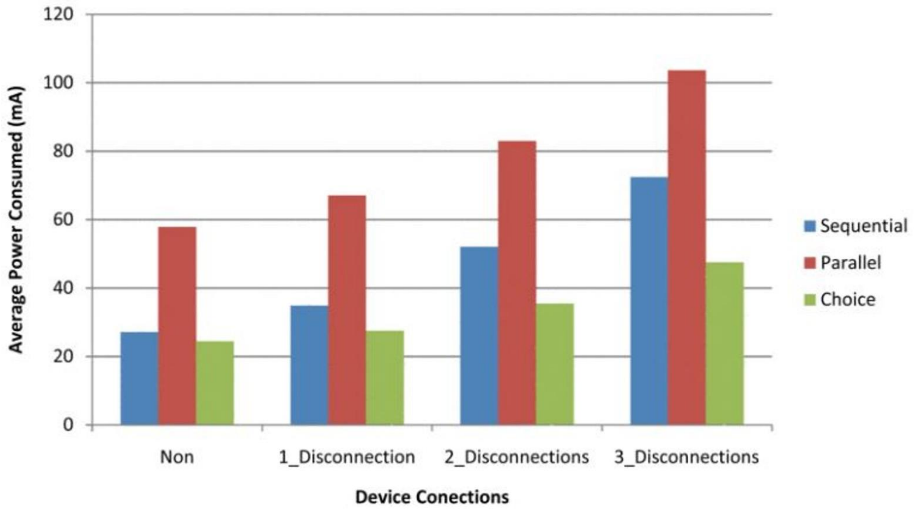


Figure 12: Average power consumption of various flow patterns [95]

Although 6LoWPAN protocols (i.e., IPv6 over Low-power Wireless Personal Area Networks) provide WSNs to use IPs to communicate with other networks, the application protocols need to be developed or modified to enable full device communication across the WSN settings. With this intention, Moritz et al. [102] introduced the use of the Devices Profile for Web Services (DPWS) as an application layer protocol in WSNs. The authors presented a SOAP binding on top of the IETF CoAP, coupled with EXI format, to reduce overhead and increase the efficient delivery of SOAP messages. However, the main shortcoming of using SOAP is that it needs heavyweight parsers, which degrades the overall performance of the mobile web server.

Doukas et al. [47] presented a generic mobile software development kit (SDK) that allows developers to easily integrate IoT protocols (i.e., WebSockets and MQTT) into their applications to communicate with the cloud-based IoT environment. The authors described the way that a mobile device could communicate with an IoT application with the generic mobile SDK called COMPOSE. The SDK enables developers to implement communication with devices and IoT services by leveraging various IoT protocols, such as MQTT, RESTful communication and WebSockets.

Previous researchers [21] introduced Pogo—a mobile telephone sensing middleware infrastructure that provides easy access to sensor data. The middleware-installed telephone is added to a shared pool of devices, and developers can remotely deploy their own executable code onto them. The authors also described the implementation of middleware and demonstrated its feasibility in a real-world Wi-Fi localisation experiment.

Perera et al. [110] presented a plugin-based IoT middleware for mobile devices: Mobile Sensor Data Processing Engine (MOSDEN). The middleware allows users to collect and process sensor data without writing any code, and supports different communication models, such as push/pull data streaming and centralised/decentralised communication.

Table 1 summarizes a comparative overview of past MWS frameworks. It shows that earlier approaches were based on providing SOAP-based MWS architectures. However, later approaches have switched their focus on the RESTful services, which is more suitable for the constrained environments because of their lightweight features [8]. Although the service architecture has been switched from the SOAP to REST, HTTP was still being used for the application protocol. Since the HTTP uses TCP as the transport layer protocol, the overhead of the protocol combination is also relatively high. To reduce the protocol overhead, a lightweight application protocol such as CoAP is needed. However, existing MWS frameworks have not addressed such features.

Year	**Author**	**Description**	**Protocols**					
			App. Layer	Trans. Layer	Local Discovery	Global Discovery	Service Description	Message Format
2003	Berger, S. et al. [15]	Custom Web service for local network	HTTP	TCP	DHCP, centralised local UDDI registry	No	WSIL	SOAP
2004	Srirama et al. [139], [138], [134]	Lightweight MH with Personal Java	HTTP	TCP		Global UDDI registry	WSDL	SOAP
2005	Gehlen and Pham [56]	P2P Web services on Java enable mobiles	HTTP	TCP	Local Service Registry (Service Broker)		WSDL	SOAP
2006	Srirama [135], Srirama et al. [140]	MH in P2P network on JXTA technology	HTTP	TCP	JXMA (JXTA) modules	JXMA (JXTA) modules	JXTA advertisements	SOAP
2008	Ou et al. [106]	Layered P2P architecture for Converged Cellular and Ad Hoc Networks	HTTP	TCP	Local service repository	Vertical tunneling model	WSDL	SOAP
2009	Kim and Lee [80]	Migration of services between devices and management of context and service directory	HTTP	TCP	Local directory manager for publish and discovery		WSDL	SOAP
2010	AlShahwan and Moessner [8]	To confirm the feasibility of providing RESTful-based Web Services from constrained mobile devices	HTTP	TCP				SOAP /REST
2011	Chang [23]	Context-aware cache prefetching strategy	HTTP	TCP	Zeroconf		WSDL	REST

2012	Paniagua and Srirama [107] [142]	MH with REST and OSGi frameworks.	HTTP	TCP	JXTA for P2P discovery. Zero-Conf /Multicast DNS for Adhoc-Network	Wide-Area DNS-SD	OWL-S/WSDL	REST
	Mohamed and Wijesekera [99]	A lightweight framework fully conforms to the REST	HTTP	TCP	Manual IP address of the MH	Manual IP address of the MH	JSON	REST
2013	Charl van der [151]	Context-aware services, a hybrid model of both Mobile P2P and JmDNS.	HTTP	TCP	Mobile P2P/ JmDNS	Mobile P2P/JmDNS	JSON	REST
	Verma and Srivastava [153]	Maintain a service directory using the XMPP	HTTP	TCP	XMPP	XMPP	Info/Query stanza of XMPP	REST
	Moritz et al. [102]	Devices Profile for Web Services (DPWS) used as an application layer protocol in WSNs	CoAP	UDP	DPWS – UDP-IP multicasting			SOAP /EXI
2015	Liyanage et al. [88]	A lightweight mobile Web service-provisioning framework for mobile sensing	CoAP	UDP	BLE	DNS/CoRE	WSDL	REST
	Doukas et al. [47]	Development of a generic Mobile SDK to integrate IoT protocols easily	HTTP	TCP				
2018	Ali et al. [6]	A comparison of SOAP and REST in mobile cloud computing	HTTP	TCP			WSDL	SOAP /REST

Table 1: Comparision of related works of mobile Web service provisioning

2.5. Using Nearby Resources (Mist Computing) in Mobile Web of Things

The classical design of IoT systems relies on remote central processing servers, and faces the issue of latency [29]. Specifically, in many real-time ubiquitous applications (e.g., augmented reality, environmental analytics and ambient assisted living), users demand rapid responses. To address this issue, researchers have introduced numerous edge computing methods. For example, the European Telecommunications Standards Institute introduced a multi-access edge computing (MEC) model [109] that enables integration between the mobile application service and the cellular network co-located virtual machine (VM) servers. Further, the fog computing architecture [19] introduces the embedded virtualisation technology in the gateways (e.g., router, switch and set-top box) of front-end IoT devices to support the data pre-processing at the edge networks. Indie fog architecture applies a consumer-as-provider model, in which small businesses and individuals offer edge computing platforms from their equipment [28].

Although edge computing methods are promising, the increasing number of IoT devices (e.g., wireless sensors, actuators, smartphones and vehicles) will overload the capability of the edge computing servers. Hence, the edge computing servers will face bottleneck issues, and latency will still arise [112]. Thus, to reduce the burden of edge computing servers, researchers have introduced the mist computing model (mist) [112, 113]. In mist, the computing moves to the extreme edge of the IoT environment, where there are a vast number of heterogeneous mobile devices and IoT devices that are capable of providing certain forms of services to assist the improvement of the computational processes needed in IoT applications [112].

As Figure 13 shows, mist servers are hosted on various objects, such as single board computers, mobile devices and embedded servers of vehicles. Essentially, mist servers (or mist nodes) are different to the regular embedded web servers (EMS) [87, 116, 72] that provide static software services to their clients. For example, an EMS can easily afford its current location as a web service via the mobile Internet with common request–response interactions. In contrast, mist nodes provide a more flexible environment in which they can execute customised program methods sent from their requesters.

Stantchev et al. [143] proposed a smart healthcare framework that uses a three-tier architecture, including a fog layer, which provides low latency. The proposed healthcare and elderly care applications are modelled using Business Process Modelling Notation, and the fog layer enhances the framework by providing low latency and location awareness. The mobile edge computing (MEC)–based Elastic Android Applications for Computation Offloading (CloudAware) [105] is a framework that offloads tasks to edge devices. Moreover, the proposed approach provides duplication and administration of the application business logic, even in untrusted and unpredictable dynamic environments. The MEC application

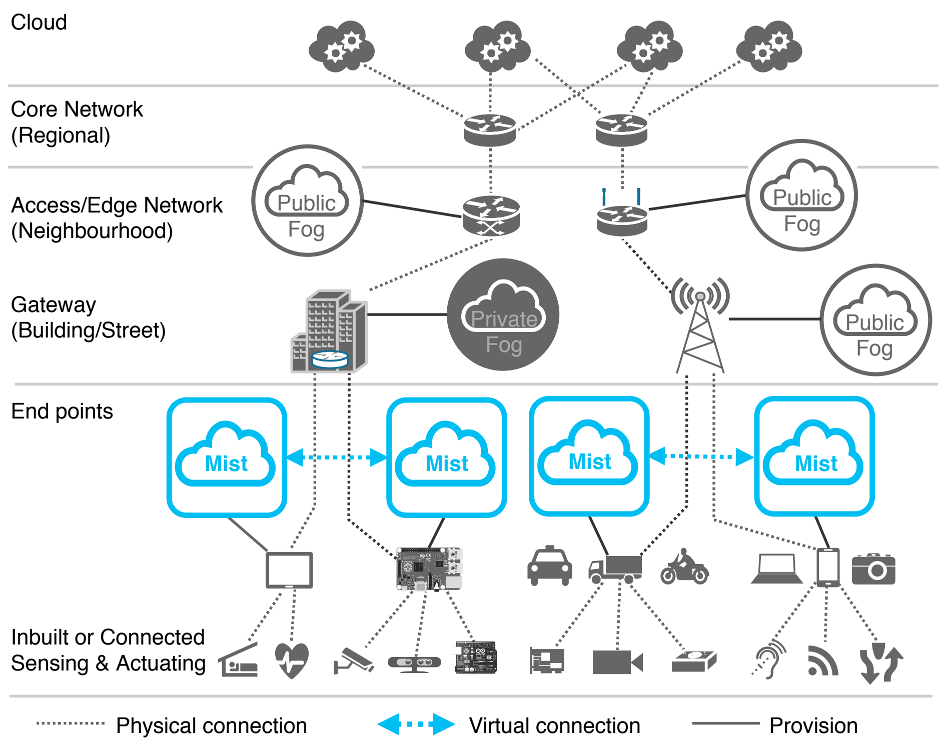


Figure 13: The role of Mist in IoT

is broken into components that simplify the offloading decision and facilitate the development of elastic and scalable applications for ad-hoc mobile clouds.

Taleb et al. [148] presented an approach called ‘Follow Me Edge’ that is also based on the concept of MEC to enhance the QoS. The framework ensures the high quality of the services by reducing the core network traffic, in which the applications/services also move with the users. Moreover, the authors demonstrated the way that high QoS can be maintained in an augmented reality use case. In this use case, a high-definition video is cached at the edge of an attractive location, and streamed to the people visiting that location on demand.

Cloudlet [120] is a framework that can be used to realise MEC and fog computing. The principal concept is to host virtual machines on the computers of local business, and provide the VM as a service to proximity-based users. Hyrax [79] is a framework that aims to apply MapReduce to the grid-computing environment formed by a group of mobile devices. However, the results for Hyrax indicate that the approach is too heavyweight.

To overcome the deployments encounters and limitations of the Cloudlets system, previous research [72] has proposed a hierarchical model of MEC-Cloudlet integrated architecture that uses the MEC system and Cloudlets infrastructure. The framework is designed for large-scale cooperative Cloudlets deployments that provide optimisation for the power consumption and delay.

The Honeybee [53] is a framework that approaches the same purpose as Hyrax. However, instead of using the heavyweight MapReduce, Honeybee applies the active work-stealing scheme, in which participants in a mobile grid-computing environment actively take the work from other nodes that have more tasks to complete. Similar works can be found in the comparison of [42].

To minimise the transfer delay between the fog and the cloud and to maximise the resource use of fog nodes, Skarlat et al. [132] presented a conceptual framework for fog resource provisioning. The authors introduced the concept of fog colonies, which are microdata centres comprising an arbitrary number of fog cells. Within a fog colony, task requests and data can be distributed and shared between the single cells, which facilitate decentralised processing at the edge of the network.

2.6. Energy-efficient Approaches in Mobile Web of Things Service Provisioning

2.6.1. Energy Efficiency in Wireless Sensor Networks

WSN is one of the main elements of IoT for sensing environments. A number of reviews of IoT [9, 11, 39, 52] have described the challenges of WSN concerning energy efficiency. Moreover, numerous energy-efficient approaches for IoT scenarios were proposed in [12, 155] to improve the lifetime of the sensor data-collection processes. The current author summarised the existing approaches into a few groups based on their common features, as follows.

Clustering-based Approaches

To increase the lifetime of the sensor nodes, many cluster head selection schemes have been developed during the last decade. Many of these were based on extending the low-energy adaptive clustering hierarchy (LEACH) [66] algorithm, which forms a cluster of wireless sensors and selecting cluster head based on signal straight. In the framework proposed in [79], the base station first collects all sensor nodes' energy and location information, and then elects the cluster heads using the fuzzy rule according to the collected fuzzy variables. In addition, an energy-aware multi-objective fuzzy-clustering algorithm for WSNs [123] provides a solution to prevent the early death of cluster heads close to the sink. The proposed fuzzy-clustering algorithm uses the three parameters of remaining energy, distance to the sink and density of the nodes to assign appropriate ranges to tentative cluster heads. The final cluster heads are then selected from them via an energy-based competition.

Scheduling Approaches

Scheduling schemes ensure that sensor nodes only function when necessary, and remain in sleep mode at all other times to reduce energy consumption. One of the conventional approaches involves dynamically scheduling the duty cycles of sensors by using sleep scheduling algorithms. The connected k neighbourhood

(CKN) [103] algorithm turns off the redundant nodes if a node has more than k-neighbours in the network. The groups of active nodes are selected periodically, and when the network does not satisfy the k-neighbours, other nodes should be in active mode until the particular node has more than k-neighbours. Correspondingly, the algorithm discussed in [162] is an extension of CKN that also considers nodes' residual energy to decide whether a node should be active or asleep. This approach enables the nodes with more power to participate actively and the nodes with less energy to participate infrequently during the sensing period.

In mobile cloud computing, a mobile client can preserve its energy by offloading its computation task to the cloud if the cost of local computation is higher than the uploading. In [157], the authors designed an adaptive energy optimal scheduling policy based on the size-controlled collaborative execution model to reduce the energy consumption on the mobile client. Moreover, the authors proposed a low-complexity threshold adaptation scheme that exploits the degenerated Monte Carlo method to estimate the threshold of locally executed data. Similarly, a collaborative execution scheduling policy presented in [164] adopted the Lagrangian relaxation-based aggregated cost algorithm. The proposed model prepares an energy-efficient task-scheduling policy to preserve the energy on the mobile device.

To overcome issues deriving from the energy limitations of the nodes and the distributed nature of WSNs, an energy-aware scheduling strategy to allocate computational tasks on small devices in WSN was presented in [37]. The proposed framework is established on the two-phase heuristic-based algorithm that initially assigns a task locally to the cluster, and, if the task cannot be completed locally, it is migrated to the most suitable node. Moreover, the focus of the scheduling strategy exploits the network lifetime by increasing the number of alive nodes with balanced energy load. In addition, efficient resource allocation for multi-hop IoT infrastructure presented in [2] introduced an energy-efficient context-aware traffic-scheduling (EE-CATS) algorithm that reduces nodes' total awake time by applying an adaptive duty cycling approach.

Correspondingly, generally in WSNs, each sensor node has a limited buffer space to hold the sensed data, and data may have overflows if not forwarded to the data collectors on time. However, the sensor nodes in different areas have different sampling rates that cause complications in designing a unique schedule to upload data from nodes to collectors. With help from mobile data collectors, a mobile element scheduling algorithm [133] enables the mobile element to visit nodes promptly to gather data to avoid data loss at the sensor nodes.

Similarly, properly scheduling the time-division multiple access (TDMA) slots in WSNs will improve the nodes' lifetime and network efficiency. The authors who presented a distributed TDMA scheduling algorithm for the IoT [84] stated that the energy and topology information of network nodes is essential for extending the network lifetime. With this in mind, they proposed an energy–topology (E-T) factor that uses the residual energy and topology information to formulate the scheduling algorithm. Further, the algorithm prioritises time slots that minimise the execution time and energy consumption of the whole network. Additionally, research on TDMA scheduling algorithms for WSNs [51] proposed two centralised scheduling algorithms—node-based scheduling and level-based scheduling—established on the colouring of the linear network.

Accordingly, another approach to enhance the energy efficiency in WSNs involves selecting a set of representative nodes that periodically provide sensor data, which will minimise the number of active nodes and messages in the network. Based on this strategy, [34] introduced an energy-efficient node-scheduling algorithm using Markov random field. In addition, [145] presented an asynchronous wake-up scheme for energy conservation in underwater acoustic sensor networks. The authors offered a combinatorial design asynchronous wake-up scheme to minimise the active duty cycle of sensor nodes.

Moreover, the research in [33] proposed a Service-oriented Node Scheduling Scheme, Energy-aware Centralised Heuristic Scheme (ECHS) and Energy-aware Distributed Heuristic Scheme (EDHS). An energy-aware benefit function used in the ECHS determines the active sensor nodes and rotate sensor nodes by periodically reconstructing the scheduling scheme. In addition, the feature improves the network performance by considering the nodes' capability of providing services and residual energy. Moreover, the EDHS provides a distributed solution by assigning one header to each service. In addition, the scheduling scheme is performed in a distributed way that, for the active node selection process, only requires local information.

2.6.2. Using Public Fog Networking Services for Energy-efficient Mobile Web of Things

Fog Computing architecture enables the IoT devices to carry out a substantial amount of storage, communication, and management, including network measurement, control, and configuration within the close vicinity of IoT devices [35], [19]. Some interesting IoT scenarios where the Fog architecture can be applied has been described in [19]. In the Connected Vehicle deployments, the Fog will be the main platform to enhance connectivity and interactions from cars to cars, cars to access points and access points to access points. Further, in the Wireless Sensor and Actuator Networks, and in the Smart Grid scenarios the Fog nodes can filter the data which is to be consumed locally, while sending the rest to the higher tiers (cloud). Fog networking is a promising approach to support energy conservation for increasing the lifetime of data acquisition from the MWoT.

To address the uncertainty of the available resources in fog networking, a Markov decision process based autonomic deep Q-learning approach has been presented in [4]. The presented approach minimizes the latency and enhances the performance of the computation offloading decisions. Also, the use of a deep-learning-based response-time-prediction to make a decision whether to offload to the nearby fog/edge node or to the cloud node has been discussed in [5]. Moreover, the authors presented a restricted Boltzmann machine learning approach to handle the uncertainty of the available resources in fog. An energy efficient Fog-assisted IoT remote health monitoring system for diabetes patients with cardiovascular diseases has been described in the work [57]. The system consists with fog-assisted smart gateways that will enhance the quality of healthcare services in which extract important information in real-time for critical decision making, push notifications for the caretakers, categorization of activities and fall detection fall detection.

A Smart Gateway with Fog Computing that can be used to pre-process or filter the IoT data has been discussed in [1]. The authors presented two types of communications with the smart gateway: single hop communication and the multi-hop communication. Both types are mainly about how the IoTs communicate with the smart gateway. In the single hop communication, the IoT nodes communicate with the smart gateway directly and in the multi-hop communication, the IoTs communicates over multiple sink-nodes to reach the smart gateway. Since the smart gateway is directly connected with the Fog network, it is possible to take advantage of using the Fog computing resources such as storage and processing power to pre-process and restructure the data before sending them to the cloud servers. In work [58] the authors presented a Fog computing based smart gateway for an efficient health monitoring system. The proposed system uses Fog-computing services, particularly low-latency and real-time rapid notifications in emergencies (which are the most critical facts in health monitoring systems), minimizing the transmitted data to the cloud, etc. The implemented prototype of the smart gateway with features of a lightweight template for ECG feature extraction, interoperability, real-time notification mechanism, etc. has already confirmed the usefulness of the system with health-monitoring applications. In summary, with the Fog networking architecture, MWoT devices can interconnect with the existing proximal FogNets providers deployed by different organizations, and use their resources for computing or networking tasks towards saving energy.

2.6.3. Service-oriented collaboration approaches for Mobile Web of Things

The Semantic Gateway as Service (SGS) [46] provides a multi-protocol proxy architecture that can perform the translation between messaging protocols, such as XMPP, CoAP and MQTT. SGS consists of three main components: multi-protocol proxy, semantic annotation service and gateway service interface. These components enable discovery of physical sensors' data and provide semantic interoperability of messages between things. A container virtualisation technology–based Gateway as Service framework [101] is designed to achieve high-energy efficiency, isolation, interoperability, fast allocation and flexibility in managing different services. Moreover, the on-demand activation containers over the socket allow dynamic allocation of various services that increase the energy efficiency. An IoT-based mobile gateway solution [118] for mobile health (m-Health) autonomously collects vital information about the user/patient—such as location, heart rate and possible fall detection—and forwards it to a caretaker. The mobile gateway architecture consists of four modules of body sensor network, mobile gateway, cloud and caretaker, which appropriately provide real-time management of actions and alarms. To provide semantic collaboration between semantic WoT applications, the Gateway-based Semantic Collaboration Service Framework [159] presented a useful protocol proxy technology that was designed to collect data from sensors that use various types of protocols. In addition, the framework uses rule-based reasoning to filter out invalid sensor data, and semantic reasoning technology to gain high-level knowledge from the data.

Recently, establishing front-end device-to-device collaboration in pervasive and mobile computing has become a popular research domain [40, 93, 156]. For example, Gateway as a Service—a cloud computing framework for the WoT presented in [156]—provides a collaborative workflow among different devices, applications. The authors proposed a three-layered cloud computing stack that improves the integration of device-level resources and information technology (IT)–enabled business processes for provisioning of intelligent IoT services.

2.6.4. Energy-efficient Service Description and Service Discovery in Mobile Web Services Provisioning

Service discovery and service description metadata (SDM) play an important role in dynamic service composition systems, where software clients can automatically identify the operations provided by the service. The following sections discuss the main service description approaches and service discovery mechanisms that are suitable for MWS provisioning.

Local Service Discovery

Local service discovery in a P2P manner is one of the major features of MWS provisioning in many scenarios [30, 140]. Different to global service discovery—which can be achieved by using a central service repository/registry—local P2P-based MWS discovery requires P2P communication protocols that are commonly available in existing mobile devices. In a classic design, the MWoT might use Wi-Fi or Bluetooth to support the P2P-based service discovery mechanism. However, traditional Wi-Fi and Bluetooth protocols are still considered heavyweight, and the BLE protocol that was designed for applications requires low power consumption and low radio duty cycle.

BLE was designed for applications that require low power consumption and low radio duty cycle. It operates in the 2.4 GHz industrial, scientific and medical (ISM) band and uses adaptive frequency hopping to avoid interferences. In BLE communication, it divides the frequency spectrum into 40 channels with centre frequencies of 2,402 MHz to 2,480 MHz, with a 2 MHz guard band for each channel. There are 37 data channels, three advertising channels that consist of 40 communication channels. With the data rate of 1 Mbps, BLE uses Gaussian frequency shift keying as the modulation scheme [45, 59].

BLE uses a lightweight attribute protocol and attribute profiles, where the maximum payload is limited to 27 octets, which is more than enough to represent sensor data, such as temperature, heart rate and humidity. To achieve the low-energy feature, the five methods supported by BLE clients and servers are PUSH, PULL, SET, BROADCAST and GET. Clients use the PULL method to retrieve attributes from the server when needed, while servers use PUSH to send data to clients. The BROADCAST methods can be used to send data to every listing client in the vicinity to minimise the bandwidth and power consumption.

Global Service Discovery

In general, traditional web discovery technology discovers resources across the Internet using a powerful search engine with the mechanism of web crawlers that browse the web periodically. However, the nature of intermittent connectivity to the web of IoT devices will create a problem when applying the traditional web discovery technologies. In addition, because of security concerns regarding IoT systems, IoT devices are generally located in domains with limited public retrieval. Therefore, we cannot apply the traditional 'pull' model of the web services discovery, and must apply a 'push' mechanism to publish the services.

To overcome the limitations in the constrained environment mentioned above, CoAP endpoints use the CoRE link format for service discovery. Two approaches can be applied to the service discovery process:

1. CoAP resource discovery
   This basic distributed approach allows a client device to send a direct query to another device to discover the hosted services. A server that is hosted services is published a list links of hosted resources using a well-known URI `"/.well-known/core"` , which is defined as a default entry point for the clients. Later, the client refers to the associated URIs to access the available resources on the server.
2. CoAP Resource Directory (RD)
   An RD [129] is similar to a conventional directory-based discovery mechanism, in which the RD stores descriptions of resources hosted on the CoAP servers, and allows clients to perform queries on those resources. To use the RD, clients and servers should know how to reach the RD. Therefore, the RD should be a well-known device, such as a gateway or DNS server.

Service Description

SDM play a vital role in MWoT service provisioning systems. The following sections discuss the main service description approaches that are suitable for MWoT service provisioning.

Web Service Description Language

Original WSDL[2] was designed to describe SOAP-based web services. However, WSDL 2.0 is capable of describing RESTful services. Following is a portion of sample WSDL 2.0 documents that can be used to describe a RESTful operation provided by the MWoT:

```
<operation ref="" whttp:method="GET"/>
<service name="LocationService"
interface="http://example.org/services"/>
<endpoint address = "coap://serv.org:5683/LocationService"/>
```

An alternative web service description, besides WSDL, is Web Application Description Language (WADL)[3], which was introduced specifically to describe RESTful services. Although WADL was not accepted as a W3C standard, it has been broadly used in practice. For example, Jetty[4] web service container supports WADL as the SDM.

Sensor Model Language (SensorML)

SensorML[5] is a metadata-based resource description language. It provides XML encodings for illustrating sensors, actuators and processes. The schema and namespace information is defined under the header of the SensorML document. All elements defined in the document mainly depend on the services and resources associated with the sensors. However, there are some required elements, such as:

- `gml:identifier`—contains a unique ID (universally unique identifier, uniform resource name, URL, etc.), which can be used to identify any service or the sensor
- `sml:outputs`—defines what is measured (such as room temperature)
- `sml:position`—defines the location of the sensor device (e.g.,`<gml:coordinates>47.8 88.56</gml:coordinates>).`

SensorML is suitable for describing MWS when the mobile host is mainly provisioning sensory data. Additionally, SensorML supports semantic description, which is not provided by WSDL 2.0 by default.

JavaScript Object Notation for Linked Data (JSON-LD)

JSON-LD [38] is a format that aims to represent the SDM in a JSON representation. It provides an application to a standards-based machine-interpretable approach that begins in one location and follows embedded links to access other pieces of linked data across the web. Moreover, JSON-LD is designed to build interoperable web services with a lightweight syntax. For instance, a location sensor data can be presented in JSON-LD format [146] as follows:

```
{
"@context":
```

[2] http://www.w3.org/TR/wsdl20/

[3]http://www.w3.org/Submission/wadl/

[4]http://eclipse.org/jetty/

[5]http://www.ogcnetwork.net/SensorML Intro

```
{
"i": "http://iot.fi/o#",
"ownerID": "i:ownerID",
"longitude": "i:longitude", "latitude": "i:latitude"
},
"@id": "i:locaSensor767",
"@type": "i:LocationSensor",
"ownerID": "Alice",
"longitude": "25.468", "latitude": "65.058"
}
```

2.6.5. Payload Compression and Encoding

Reducing the size of payload in the data transmission phase is one of the main strategies to improve the performance and energy efficiency in MWS provisioning. Most of the embedded web services utilize XML to encode the payload that is adding heavyweight and may not a good option for mobile devices.

Binary XML (to encode dense numeric data) and JSON are common options in past works [136]. However, the recent W3C standard—Efficient XML Interchange (EXI) is capable of outperforming Binary XML and JSON in reducing the data size. In the EXI compression, the XML document is encoded into a binary format that improves encoding/decoding performance and significantly reduces bandwidth requirements [22]. Table 2 [125] shows the compression efficiency of XML, EXI, BXML, and Fast Infoset (https://www.noemax.com/fastinfoset/)encoding techniques. There are three typical sensor markup schemes, namely 1) Resource Description Format (RDF), 2) ZigBee Smart Energy and 3) Open Geospatial Consortium (OGC) SensorML have been used to compare the compression efficiency. Overall EXI has a high compression efficiency when compared to other compression schemes [125].

Encoding	Complexity	RDF	Smart Energy	SensorML
XML	Medium	206	409	300
EXI*	Low	6 (3%)	13 (3%)	57 (19%)
BXML	Medium	177 (86%)	210 (51%)	177 (59%)
Fast Infoset	Medium	143 (69%)	200 (49%)	185 (62%)

Table 2: Comparison of the related compression schemes (size of encoded objects in Bytes) [125]. (*Strict schema-informed mode with bit alignment)

2.7. Summary

This chapter has presented a literature review of the state-of-the-art developments in the MWoT domain, specifically related to system architectures, protocols, communication mechanisms and so forth. At the beginning of the review, the author discussed some lightweight operating systems, which are mainly designed for the energy preservation of sensor networks. When considering the MWoT, the energy conservation of the mobile device plays a key role, and the discussion also included the features of some energy-efficient communication protocols. The author then discussed the recent developments of the WoT architecture and the W3C standardisation of the WoT.

This chapter then presented a review of the lightweight middleware for the MWoT, including the available applications, a comparison of the existing frameworks, the use of nearby resources in the MWoT based on the mist computing approach, and a discussion of some existing approaches. The chapter also summarised numerous energy-efficient approaches for WSN scenarios into a few groups based on their common features. Finally, the chapter discussed the use of public fog networking services for energy-efficient MWoT, including the different studies performed and a review of energy-efficient service description and discovery in the MWoT.

In summary, this chapter has explored the theoretical and technological aspects of the MWoT that provided the necessary fundamental background of this thesis.

3. LIGHTWEIGHT MOBILE WEB SERVICE PROVISIONING FOR THE INTERNET OF THINGS MEDIATION

3.1. Introduction

The capabilities of recent mobile devices such as smart phones and tablets have been steadily enhanced with faster processing power, larger and higher resolution display screens, greater memory, and enhanced power saving mechanisms. Additionally, accessing high speed mobile Internet (e.g., 3G/4G or LTE[1]) with mobile devices has become a common phenomenon in most urban areas of the world. Cisco Visual Networking Index Forecast Project predicts that, by 2018, the population of mobile device users will reach around 5 billion together with more than 10 billion mobile devices connecting to the Internet[2].

Confluence of these mobile developments with the evolution of service-oriented architecture technologies have led to the mobile Web services, where the mobile terminals are being used as both Web service clients and providers (which is termed-Mobile Hosts [139]). Mobile Web service provisioning has been utilised in numerous fields such as Location Based Services (LBS) [163], Mobile Health Services [78], Mobile Social Networking applications [161], etc.

Although recent smartphones are quite capable in terms of data transmission speed and computation power, when they are utilised for providing MWS, the challenge of resource constraints still exists. The high frequent usage of the high performance multi-core mobile CPU and the high speed 3G/4G mobile Internet data transmission will quickly drain the battery power of the mobile device. Therefore, in past years, several lightweight mobile Web service provisioning approaches such as lightweight HTTP based RESTful Web service frame work for mobile devices [99], optimal Data Serialization Formats for energy efficiency on mobile devices [147] have been proposed to address the resource intensive issues in MWS.

In general, there are two trends of approaches:

1. Reducing the complexity of the messaging. e.g., utilising REST based service provisioning instead of SOAP;
2. Utilising external resources to enhance the overall performance. e.g., offloading the complex computational tasks to static Cloud [30] or to the Mobile Ad Hoc Cloud [31] and also using optimistic collaboration and scheduling to reduce the consumed bandwidth in order to serve energy [26], [25].

However, when considering MWS based sensing provisioning systems, they still have some other challenges that need to be addressed in order to increase their

[1]http://www.3gpp.org/technologies/keywords-acronyms/98-lte

[2]http://newsroom.cisco.com/release/1340551

efficiency. For example, when considering the mobile sensing scenarios, instead of providing the information regularly based on the requests, sometimes it may be ideal to store the information on the cloud and provide access to the data, for the further requests. This way, the communications latencies can be significantly reduced, if there are server bunch of clients requesting the same information.

In addition, there should be a proper way to handle the conflicts and compatibility of services. Mobile devices have limited sensing components and they may not be able to operate concurrently. Conversely, some services may be able to operate at the same time, even though they are using the same sensing components. For example, video-based sensing and image-based sensing services both use camera component and may operate at the same time depending on the specification of the devices.

In such cases, how does the Mobile Web server manage the services and provide timely service publishing?

Similarly, in the case of real-time or a periodical sensing service operation, the executed service can affect the availability of other services.

How does the Mobile Web server measure or prioritise the availability of the services?

To address these issues, we extend the work [88] towards presenting a lightweight mobile Web service provisioning framework based on integrating a number of lightweight protocols including Constrained Application Protocol, Bluetooth Low Energy and Efficient XML Interchange. We also have added a service scheduling feature to address the discussed challenges and to provide uninterrupted service provisioning while maintaining the basic functionality of the mobile device.

3.2. Overview of Architecture

Figure 14 illustrates the architecture of the proposed Mobile Web service provisioning. In this environment, Mobile Web server represents the Web service provider that is capable of providing various data to its clients. For example, the Mobile Web server can collect sensory data from its surrounding sensor devices - SN1 and SN2 - and interpret the data to the useful information, then provide the interpreted data via its Web service mechanism. The proposed Mobile Web server has adopted some Web of Things recommendations that enable it to interact with any entity on the Web of Things. For instance, utilizing the EXI encoding scheme to compress message payload, CoAP as an application protocol and support RESTful methods such as GET, POST, PUT and DELETE to interact with the resources can be mentioned. Since we have embedded the WoT requirements to the Mobile Web server and as discussed in Introduction Chapter, we term this Mobile Web server as a Mobile WoT server.

The MWoT server supports two types of service discovery: *global service discovery* and *local service discovery*. The global service discovery is realised by publishing its SDM (e.g., WSDL, WADL or SensorML) to a global service

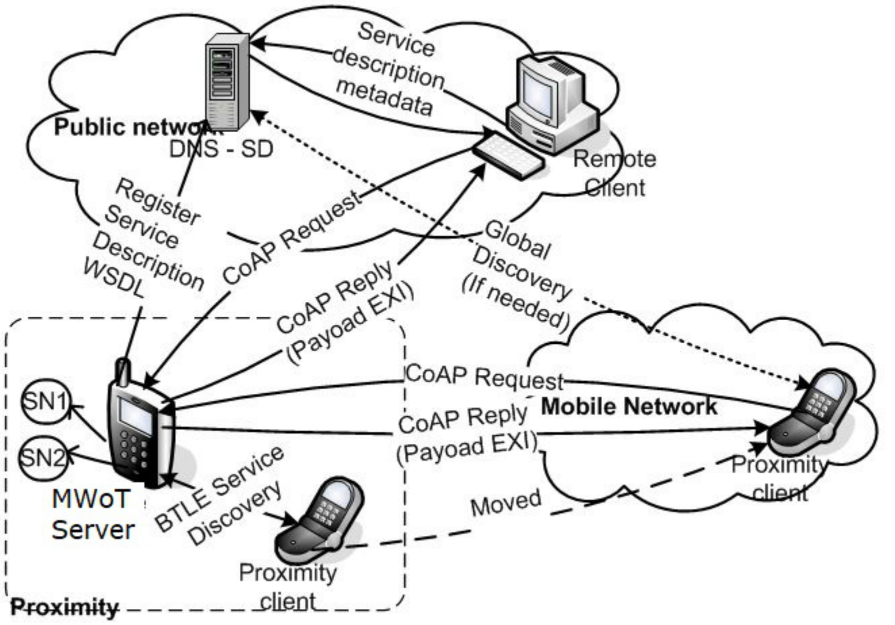


Figure 14: MWoT service provisioning framework

registry (e.g., DNS-SD). A remote client that knows the location of the service registry can discover the MWoT server by requesting the service registry. Afterwards, the remote client can perform the regular service request to the MWoT server by sending the CoAP service requests. The local service discovery is based on BLE. As Figure 14 shows, a client can discover the MWoT server in proximal range via BLE. The MWoT server will provide certain information via BLE to help the client to identify the communication requirement. If the client moves out of the BLE range with the MWoT server, it can still communicate with the MWoT server via a mobile Internet connection. The MWoT server has been assigned a public static IP address from the mobile Internet service provider (e.g. Estonian TELE2 and EMT both provide this service). Alternatively, if the public IP address allocation is not available, the MWoT server can utilise proxy services [71], or hole-punching/relaying [141].

3.2.1. Sensing Service Provisioning

The MWoT server can provide services based on time or resource allowance. For example, the user of the MWoT server can set 5 hours as the service provisioning period and 50% as the maximum battery allowance. Hence, whether the 5 hours period has past or the battery has consumed 50%, the service provisioning will be terminated, and the corresponding service publishing will be withdrawn from the discovery server. MWoT service framework supports three types of sensing services such as:

- ***One-time Sensing*** represents a generic Web service request/response operation that triggers corresponding sensing components to retrieve data and delivering the data to either the requester or specific end-points.

- ***Real-time Sensing*** represents the streaming-based sensing service. The MWoT server performs the sensing activity continuously and provide the data to the client concurrently.
- ***Periodical Sensing*** represents the activity that is triggered for each timestamp based on the request. For example, the request message may describe that for every 30 minutes; the MWoT server will perform the sensing activity once, and provide the data to a specific endpoint based on the address described in the request message.

3.2.2. Basic Protocol Stack

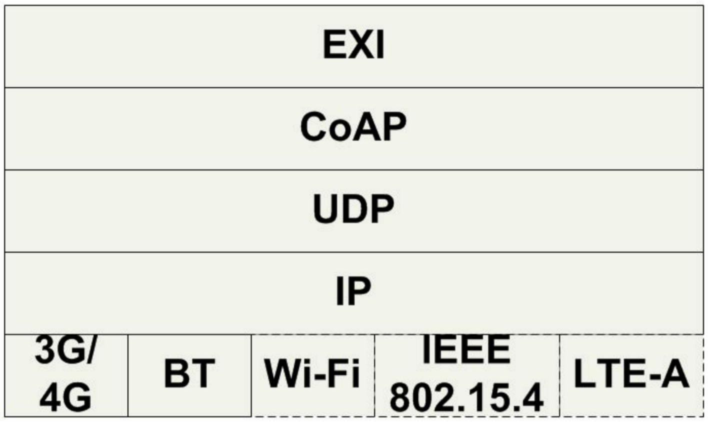


Figure 15: Protocol stack of the mobile Web server

Figure 15 illustrates the conceptual design of the protocol stack used in the proposed MWoT server framework. The proposed framework is mainly based on lightweight protocols to make the process simple. In the lower layer of the protocol stack, the mobile MWoT service framework can utilise numerous common protocols such as the 3G/4G mobile Internet (for global service interaction), Wi-Fi/Wi-Fi Direct, Bluetooth (BT)/BLE, IEEE 802.15.4 (e.g., ZigBee), or LTE-A (LTE Direct) for the local service interaction.

Unlike traditional web servers based on HTTP, our application layer utilises CoAP that provides a solution with a compact header size, simple, lightweight, RESTful message exchange in between the MWoT server and clients over constrained network.

As mentioned in the earlier, the proposed MWoT server framework also utilises EXI to compress the size of the message payload. Since the design of the framework is for provisioning sensory data in IoT scenarios, service description is very important because it facilitates to communicate smart object over the network. We proposed to use SensorML based resource description which provides XML encoding and the semantic description of sensory data.

3.2.3. Components of the MWoT framework

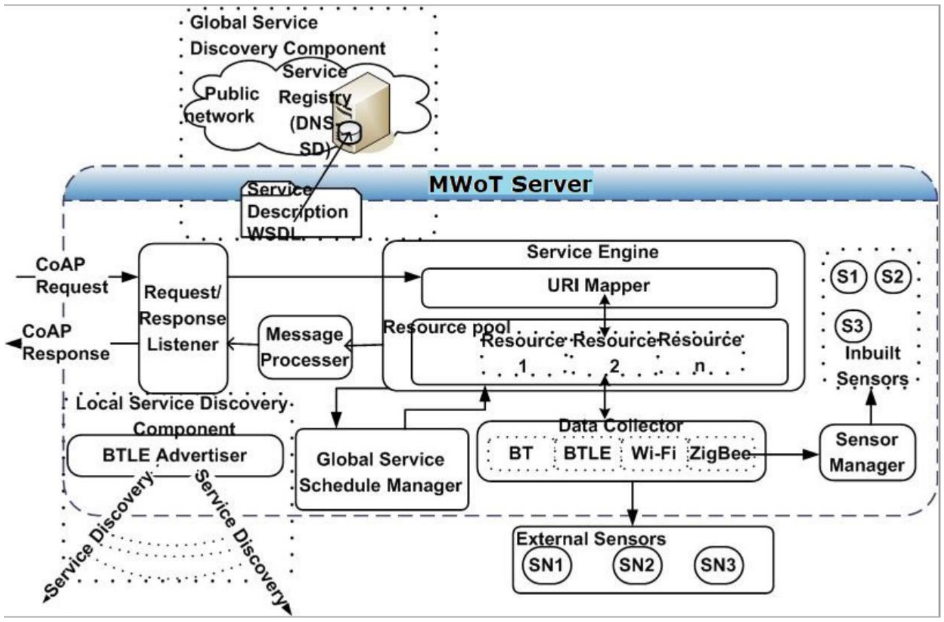


Figure 16: Main components of MWoT server

Figure 16 illustrates the component architecture of the proposed MWoT server framework. It consists of following main components.

Request/Response Listener

It is the first contact point for the clients to the the MWoT server. Once the server is started, it listens on port 5683 which is reserved for the CoAP services. Clients will get the contact details of the server with the service description from the local/global service discovery components that will be discussed later.

Service Engine

This is the core module of the the MWoT server that coordinates all the parts of the parts of the framework. Main tasks include analysing requests from the clients, processing and executing relevant methods to get data from a particular resource, and then sends data back to the clients with the response messages. In order to handle services, the service engine has two main sub components:

- ***URI Mapper***—Once the client request is received from the Request/Response Listener, URI Mapper analyses the URI to fetch the service information which is requested by the client. The requested service indicates what kind of data or service it requires and the URI Mapper locates the corresponding resource within the engine to fetch the data.
- ***Resource Pool***—Service engine maintains a collection of resources that has context data about particular Web services. Resource Pool has an instance

for each CoAP resource which will execute upon the client request. The Resource Pool also communicates with the Data Collector to fetch data from the data providers for particular resources.

Data Collector

Data collector is responsible for collecting data from available resource providers. Such providers can be the inbuilt sensor components of the MWoT server device, external environmental sensory service providers or spatial data from mobile social network in proximity. The Service Engine selects the type of Data Collector that can retrieve data from the sensors.

Message Processer

To reduce the size of the CoAP message the Message Processor processes the payload of the messages used in the communications. The proposed MWoT server framework utilises EXI-formatted binary scheme. EXI Parser gets data from the Service Engine, compresses it, and forwards to the service engine which will wrap it to the CoAP message.

Sensor Manager (SM)

It is responsible to communicate with the inbuilt sensors. One of the main fact which consume more energy is applications that are reading data from inbuilt sensors of a mobile device. When the client request is related to inbuilt sensory data, the Sensor Manager collaborates with the operating system to activate the sensor components, fetch the data and deactivate the sensor. Sensor manager only activates the sensor components when it is necessary and helps to minimise the energy consumption of the MWoT device.

Local Service Discovery Component

When the client device locates itself in close proximity to the server, it can receive the BLE advertisement broadcasted by the the MWoT server which is more energy efficient than other types of discovery mechanisms. After the BLE connection is established, the client can explore the basic services that are provided by the server.

Global Service Discovery Component

To enable the mobile Web services provisioning across other networks, the MWoT server registers its service description with the global DNS server. In RESTfull architecture, all Web services publish with associated URIs that can be discovered by the clients.

Global Service Schedule Manager

The MWoT server can provide services based on available time and resources. The main task of the service scheduler manager is providing a schedule for the service provisioning which is defined as below:

Definition 3.1. *(Global Sensing Service Schedule - GS)*

GS is defined as a tuple ($\mathscr{T}$, λ) *where:*

- *$\mathscr{T}$ is a set of timestamps for service provisioning period. Each timestamp is scheduled as 1 minute. i.e. for a 1 hour service provisioning, $|\mathscr{T}| = 60$.*
- *$\lambda : \mathscr{T} \rightarrow \mathscr{S}$ maps timestamps with the scheduled sensing service executions.*

MWoT server supports three main types of sensing services as described earlier in this section. The sensing service request can be performed in two forms as simple or complex. To enhance the sensing service provisioning, a model of the sensing service schedule is defined and described in the following section.

Sensing Service Scheduler Model

In order to optimise the service provisioning, use of the sensing service pool in service schedule and defined as below:

Definition 3.2. *(Sensing Services Pool- SSP)*

Sensing service pool describes the information of sensing services provided by the MWoT server. It is defined as a tuple $(\mathscr{S}, \varsigma, \kappa, \varepsilon)$ where:

- *$\mathscr{S}$ is a set of sensing services.*
- *$\varsigma : \mathscr{S} \rightarrow \mathscr{R}$ maps sensing services to sensing components (e.g.,GPS sensor,accelerometer camera, audio recorder, network signal browser etc.).*
- *$\kappa : \mathscr{S} \rightarrow \mathscr{S}$ maps sensing services to conflict services.*
- *$\varepsilon : \mathscr{S} \rightarrow E$ maps sensing services to the system resource usage sets (e.g. CPU, RAM, network transmission bandwidth usage etc.).*

Example 3.1. *(Conflict Service)*

Let s_1 and s_2 be two services. Let $\mathscr{T}' \subseteq \mathscr{T}$. Assume s_1 has been requested by a real-time sensing request, which has its timestamp period within $\mathscr{T}'$, and s_2 is not requested by any client. Suppose $s_2 \in \kappa(s_1)$, then s_2 will be marked as unavailable during $\mathscr{T}'$.

Sensing Service Request Processing

If the request is a simple service request that involves only a one-time service invocation that has only one activity. Otherwise, the request will be handled as the description provided in the request itself. It is defined as below:

Definition 3.3. *(Request Processing - RP)*

RP is defined as a tuple $(\mathscr{N}, \mathscr{F}, \tau, \mathscr{P})$ where:

- *$\mathscr{N}$ is a set of nodes.*
- *$\mathscr{F} \subseteq \mathscr{N} \times \mathscr{N}$ is a set of flow relation rule.*
- *$\tau : \mathscr{N} \rightarrow \mathscr{Y}$ maps nodes to node types. A node type can be an activity, a gateway, an event, a sub-process etc. An activity node that involves sensor service is marked as sType.*
- *$\mathscr{P} = \{p_1, ..., p_n\}, n \in \mathbb{N}$, is a set of work schedule plan that identifies when and how long the process needs to be performed. Each $p \in P$ is defined as a tuple (st, et), which correspond to start time (st) and end time (et).*

Example 3.2. *(Work Schedule Plan Handling)*

Suppose a XR contains $\mathscr{P} = \{p_1, p_2\}$. Let (st_1, et_1) be the start time and end time of p_1. $st_1 = 14:00$ and $et_1 = 14:10$, which denotes a 10 minutes sensing task. Schedule Manager will model p_1 as a set of timestamp that consists of $\{14:00, 14:01, 14:02, 14:03, ..., 14:10\}$ when it is processing the request.

Schedule Manager analyses the request type based on the elements in $\mathscr{P}$. It is based on the following rules:

- One-time request: $|\mathscr{P}| = 0$.
- Real-time request: $|\mathscr{P}| = 1$, the end time—*et* of $p_1 \in \mathscr{P}$ is not *null*.
- Periodical request: $|\mathscr{P}| \geq 1, \forall p \in \mathscr{P} : et_p \neq null \wedge st_p \neq null$.

For any request that exceed the scope defined above, is considered as an invalid (i.e. insufficient parameter) request, which will trigger an error.

One-time Request Processing

A one-time request involves $1...\mathbb{N}$ sensing service invocation (services of *sType* nodes), which is denoted by $\mathscr{S}^{XR}$, $\mathscr{S}^{XR} = \{s_i^{XR} | 1 \leq i \leq \mathbb{N}\}$. Let $t_x \in \mathscr{T}$, and $\mathscr{S}_x = \lambda(t_x)$ a set of executed services at t_x, and $s_x \cong now$, in which *now* denotes the current system time. Let $\mathscr{S}' = \mathscr{S}^{XR} \bigcap \mathscr{S}_x$, if $|\mathscr{S}'| > 0$ then $\forall s \in \mathscr{S}'$, sensing service—$s$ will be assigned for executing the activity. Let $\mathscr{S}'' = \dfrac{\mathscr{S}^{XR}}{\mathscr{S}_x}$, For each $s \in \mathscr{S}''$, the corresponding task will be replaced from the original sensing service invocation task that will forward to the new service which retrieves sensing data which has already been gathered during the previous real-time or periodical requests.

Real-time and Periodical Request Processing

There are two main differences in between periodical sensing and real-time sensing. First, in periodical request, the request processing can be executed in a number of periods but, the request executes only once in the real-time sensing. Second, in periodical request, the request can set a specific start time, while, the real-time sensing just executes.

Service Availability Measurement and Schedule Publishing

Schedule Manager updates the service availability information when it progresses a new request. The service availability is influenced by two factors: service conflicts and system resource allowance (i.e. CPU, RAM, network bandwidth etc.).

Let $\mathscr{S}^{t_x}$ be a set of scheduled service executions at timestamp $t_x \in \mathscr{T}$, $\mathscr{S}^{t_x} = \{s_m^{t_x} | 1 \leq m \leq \mathbb{N}\}$. $\kappa(s_m^{t_x})$ denotes a set of conflict services of $s_m^{t_x}$. Hence, a set of conflict services at timestamp—t_x (denoted by $\mathscr{K}^{t_x}$) will be:

$$\mathscr{K}^{t_x} = \bigcup_{m \in |\mathscr{S}^{t_x}|} \kappa(s_m^{t_x}), \tag{3.1}$$

and the available services at timestamp t_x (denoted by $\overline{\mathscr{S}^{t_x}}$) will be:

$$\overline{\mathscr{S}^{t_x}} = \frac{\mathscr{S}}{\mathscr{K}^{t_x}} \tag{3.2}$$

The above process has only filtered the services based on conflicts. Following is the process that considers the system resource availability.

Let $E^{sys} = \{e_o^{sys} | 1 \leq o \leq \mathbb{N}\}$, be a set of the available system resources (Note that available system resources are different to the hardware specification of the mobile device. User can set the availability in percentage to avoid the service provisioning affecting to the normal use of the mobile device).

Let $\mathscr{S}^{t_x}$ be a set of sensing services assigned at $t_x \in \mathscr{T}$. For each service—$s_z \in \mathscr{S}^{t_x}$, its system resource consumption is found in $\varepsilon(s_z)$. Let $E^{s_z} = \varepsilon(s_z)$ in which $E^{s_z} = \{e_o^{s_z} | 1 \leq o \leq \mathbb{N}\}$ in which the system resource denoted by $e_o^{s_z}$ and e_o^{sys} are the same, and let $ve_o^{s_z}$ be the usage value of $e_o^{s_z}$ and ve_o^{sys} be the remaining value of e_o^{sys}. For each t_x, the ve_o^{sys} after assigning $\mathscr{S}^{t_x}$ (denoted by $ve_o^{t_x}$) will be:

$$ve_o^{t_x} = ve_o^{sys} - \sum_{s_z \in \mathscr{S}^{t_x}} ve_o^{s_z} \tag{3.3}$$

Let $E^{t_x} = \{ve_o^{t_x}\}$. Referring to previous result, $\overline{\mathscr{S}^{t_x}}$ is a set of services that has been identified as available at t_x. For a service—$s_y \in \overline{\mathscr{S}^{t_x}}$, let E^{s_y} be the system resource usage required by the service. If $\exists e_o^{s_y} \in E^{s_y}$, such that $ve_o^{s_y} > ve_o^{t_x}$, which indicates that the service–s_y requires higher usage value than the actual available resource value. Hence, the s_y is considered as unavailable at the timestamp—t_x.

3.3. Summary

In this chapter, the author has presented a lightweight mobile Web service provisioning framework for the resource-constrained environment. The author discussed the requirements of integrating a number of lightweight protocols, including CoAP, BLE, and EXI in energy efficient mobile sensing frameworks. Moreover, it described the features of service scheduling and management of conflict services that enables mobile users to participate in different mobile phone sensing systems without affecting much of their hardware resources. Details of the prototype implementations and the evaluation results of the proposed lightweight MWoT framework will be discussed in Chapter 6.2

4. ADAPTIVE MOBILE WEB OF THINGS SERVER FRAMEWORK FOR MIST COMPUTING IN THE INTERNET OF THINGS

4.1. Introduction

The classic design of IoT systems which relies on remote central processing servers, face the latency issue [29] especially in many real-time ubiquitous applications such as augmented reality, traffic analytic and ambient assisted living.

Recently, Fog Computing models have been introduced to overcome the latency issue by utilising the proximity-based computational resources such as the computers co-located with the cellular base station, grid router devices or computers in local business. However, the increasing users of Fog Computing servers will cause bottleneck issues, and consequently the latency issue to arise again.

To address the bottleneck issues in the Fog computing model, researchers have introduced Mist computing model where the computing moves to the extreme edge of the IoT environment.

This chapter extends the MWoT server framework introduced in the last chapter and applies it in the mist-computing environment towards reducing the burden of fog servers. The author proposes a service-oriented mobile-embedded Platform as a Service (mePaaS) framework enables the edge IoT devices to provide a platform that allows requesters to deploy and execute their own program models. However, in order to successfully achieve the goal, the MWoT server needs an adaptive service scheduling scheme that considers runtime context factors such as CPU load, RAM availability, etc. Accordingly, the framework supports resource-aware autonomous service configuration that can manage the availability of the functions provided by the Mist node based on the dynamically changing hardware resource availability. Additionally, the framework also supports task distribution among a group of Mist nodes. The prototype has been tested and performance evaluated on the real world devices.

4.2. Proposed Framework

4.2.1. Architecture overview

The proposed mePaaS framework enables MWoT devices to provide a flexible way of sharing their computational and networking mechanisms as services. The core of mePaaS framework is based on Enterprise Service Bus (ESB) [116] architecture. ESB is a software infrastructure that can easily connect resources by combining and assembling services to achieve a loosely-coupled Service-Oriented Architecture (SOA). mePaaS utilises a plugin module-based approach to mediate native computational and networking components to services that can be invoked by the process execution engine. Requester of mePaaS can send a request package, which consists of input parameters and the flow of processes described in the standard workflow model (e.g.BPMN `http://www.bpmn.org`) with customised algorithm defined in the supported script language of the process execution engine. mePaaS can execute the workflow that facilitates the available service modules to complete the tasks from the requester. Figure 17 illustrates the main elements of mePaaS framework and the elements are described below.

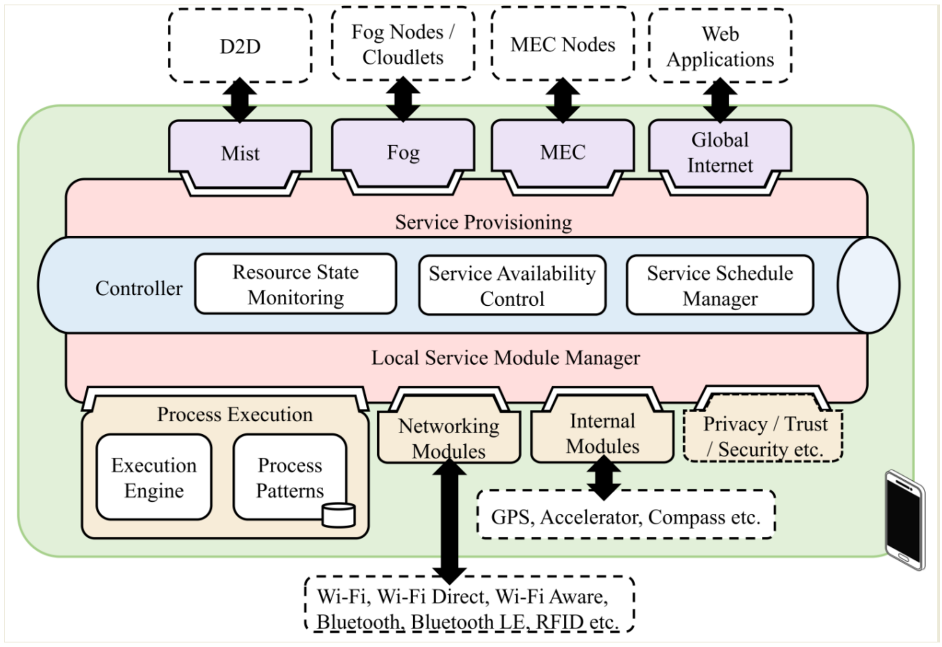


Figure 17: mePaaS architecture

Controller

Controller is the core element of mePaaS framework. Its main tasks include mediating and managing the computational and networking modules in order to coordinate with the service provisioning component. Moreover, the controller analyses the requests from the clients, and updates the service description according to the availability of resources. It includes the following important components:

- ***Service Availability Controller*** The Service Availability Controller updates the service availability information when it progresses a new request. Since each service module execution consumes a certain amount of hardware and other resources of the device, the service availability controller notifies the Controller about the conflicts of services and the overhead usage of the system resource (i.e., CPU, RAM, network bandwidth, etc.). For example, let Req1 and Req2 are two requests that use the camera for a sensing task. Assume Req1 has been requested for 20 second video capturing. If Req2 also wanted a picture from the camera at the same time, the Req2 will be marked as a conflicting service and is available only after 20 seconds. Under this situation, the Controller will notify the Service Provisioning component to disable the corresponding service (e.g. image-based sensing) from the service description until the corresponding resources are released.

- ***Resource state monitoring –*** This component is responsible for monitoring the resource usages continuously. For example, monitoring the device hardware status, CPU load percentage, RAM usage, incoming and outgoing network transmission status, etc. Based on the information arrived from this component, the controller can autonomously make a decision about which service modules are available based on the required usage of resources. The decision-making scheme is described in the next section.
- ***Service Schedule Manager –*** This is the component which can predict the availability of resources in the temporal domain. More details of this component are discussed in the later section.

Service Provisioning (SP)

SP is the component that handles which service modules can be included in the SDM. mePaaS can publish or advertise its SDM in different networks, depending on the application use case. In general, the SDM follows the W3C's recommendation for RESTful services in the WoT, in which SDM is described based on JSON-LD format. SP associates with different Service Provisioning Adaptors (SPA) such as Mist, Fog, MEC, the Web, to publish or advertise mePaaS's SDM. Also, the SPAs are responsible for handling the incoming request packages from their field. For example, Mist SPAs are responsible for advertising SDM on the D2D network (e.g. Wi-Fi Direct, Physical Web `https://google.github.io/physical-web`) and handling the requests from D2D network. Note that, distinguished from mobile ad-hoc network, the D2D utilises existing infrastructure for the communication within a 1-hop range [85].

Local Service Module Manager (LSMM)

LSMM is responsible to launch, terminate and manage the local service modules of mePaaS. Local service modules can be seen as independent software components that can be installed as plugins of mePaaS. Initially, mePaaS should have at least one process execution module and a number of corresponding modules that can utilise the inbuilt functions of the device (e.g. access GPS data). mePaaS provides a flexible way for developers or users to add more modules for supporting different needs. For example, a user can install additional modules for performing semantic sensory data reasoning. LSMM will inform the Controller about newly installed module, along with the corresponding descriptions. The controller will pass the information to SP to include the module as a new service in SDM.

Process Execution

This module consists of the following main elements:

- ***Execution Engine –*** is software to execute the workflow which is included in the request package. The Execution Engine needs to associate with

LSMM in order to invoke the corresponding service modules involved in the workflow tasks.

- ***Process Patterns –*** manage a number of predefined workflow patterns. The pre-defined patterns can be used to replace the inexcusable tasks in the workflow as substitutions. When it receives a request that contains the goal of the process, a corresponding abstract workflow model will be executed. A flow relation pattern defines the structure of a set of workflow nodes. The definition of abstract workflow model and approach will be described in a later section.

Networking Modules

These modules represent the service modules that can invoke the functions of networking requests (e.g. Sending HTTP or CoAP requests, sending Bluetooth GET request, retrieving Bluetooth Beacon's data or reading RFID or data from ISO/IEC 20248 devices etc.). These modules can also be used to fetch the SDM of the other service providers in proximity.

Internal Modules

Represents the service modules that only involve the functions from inbuilt hardware resources and sensors such as GPS, accelerometer, compass, brightness etc.

Privacy, Trust, Security, etc.

are the modules that involve the management of privacy and service level agreement, the trustworthiness and quality of service provisioning, cryptography and other security involved mechanisms. the author has studied them earlier in other contexts such as service discovery [24], and they will be integrated later.

4.2.2. Self-configured service provisioning

Mist node publishes the available services in SDM that the clients will receive in the JSON-LD format. Due to the dynamic nature of the resource usage of service modules, the availability of the service provisioning is also unpredictable. For example, if the Mist node is currently serving a continuous data streaming task, then it is unlikely to serve a new request that also requires a heavy bandwidth usage. Hence the corresponding service may need to be disabled from SDM since the Mist node cannot handle any more such kind of requests. In order to dynamically update the SDM, the author proposes the service scheduling scheme. First, the author explains the terminologies used in the scheme. As described earlier, the Mist node-handled process is modelled as a workflow. Here, the author refers to the terms described in [150] where a task that is to be accomplished is called a*work item*. A work item in mePaaS is executed by a *service module*.

Definition 4.1. *(Installed Service Module Collection) —is defined as a tuple $< S, \beta >$ where:*

- $S = \{s_l : 1 \leq l \leq \mathbb{N}\}$ is a set of service modules. Each $s_l \in S$ is defined as a tuple $< ID, type, status >$ corresponding to *identification*, *type of the service*

module (e.g. CPU intensive, bandwidth intensive etc.) and the *availability status*.
- $\beta : S \to \mathscr{U}$ is a function that maps the service modules to the required hardware usages.

Definition 4.2. *(Scheduled Work Items) — is defined as a tuple* $< \mathscr{W}, F, \zeta, \gamma, \delta >$ *where:*
- $\mathscr{W}$ is a finite set of work items. $\mathscr{W} = \{\omega_1, \omega_2, ..., \omega_n\}$, $n \in \mathbb{N}$.
- $F \subseteq \mathscr{W} \times \mathscr{W}$ is a set of flow relation rules.
- $\zeta : \mathscr{W} \to Z$ is a function that maps work items to status.
- $\gamma : \mathscr{W} \to \Gamma$ is a function that maps work items to start times.
- $\delta : \mathscr{W} \to D$ is a function that maps work items to estimated execution duration.

Let $\bullet\omega = \{\omega | (\omega, \upsilon) \in F\}$ *be the pre-set of* ω*,* $\omega\bullet = \{\omega | (\upsilon, \omega) \in F\}$ *be the post-set of* ω*.*

Definition 4.3 ((Device Hardware Usages (H^{Glo}))).

$H^{Glo} = \{h_1^{Glo}, h_2^{Glo}, ..., h_n^{Glo}\}, n \in \mathbb{N}$. Each $h^{Glo} \in H^{Glo}$ is defined as a tuple $< ID^{Glo}$, cu^{Glo}, $xu^{Glo} >$ where:
- ID^{Glo} denotes the identification of the hardware.
- cu^{Glo} denote the current assigned usage of the hardware based on the scheulded work items.
- xu^{Glo} denotes the maximum availability of the hardware.

Initially, we can apply Algorithm 4.1 in which the Service Provisioning (SP) component can decide whether to keep or remove the services from SDM based on the availability of hardware resources.

Algorithm 4.1 Pseudocode for our algorithm

```
for $s \in S$ do
    $hardwareUsageSet \leftarrow \beta(s)$
    for $hus \in hardwareUsageSet$ do
        for $h^{Glo} \in H^{Glo}$ do
            if $ID_{hus} \equiv ID_{h^{Glo}} \wedge (value_{hus} + cu_h^{Glo}) > xu_h^{Glo})$ then
                add $s$ to $removeList$
            end if
        end for
    end for
end for
```

However, Algorithm 4.1 does not consider that the hardware usage relative to the temporal space. For example, there may be a process that will terminate and release the resources in very near future. To overcome this limitation, as a future development of our previous work [90], especially for the complex type of services, the author combined a Service Schedule Manager into the Controller to enhance the decision making process and to ensure the unnecessary removing of services. For example, there may be a service request that takes some images from the surrounding and uploads them to a server periodically. Assume that the request has started at 10.00 am and continues in every 10 minutes as at 10.10, 10.20,..., until 11.00 am. In that situation, availability of the camera resource is almost free except only at some occasions. Moreover, the system already knows the availability of resources in advance and the framework can make optimal decisions accordingly. The following section describes the service schedule that optimises the service provisioning.

4.2.3. Service Scheduler

The following section describes a detailed overview of the service scheduling that enhances the service provisioning. The service scheduler combines the available resources with the domain in advance to minimise the service conflicts.

Definition 4.4. *(Local Services Pool- LSP) — LSP describes the information about local services provided by the mist node. It is defined as a* $(\mathscr{S}, \varsigma, \kappa, \varepsilon)$ *where:*

- $\mathscr{S}$ is a set of service modules.
- $\varsigma : \mathscr{S} \rightarrow \mathscr{R}$ maps services to hardware components (e.g. GPS sensor, accelerometer, camera, audio recorder, network signal browser etc.).
- $\kappa : \mathscr{S} \rightarrow \mathscr{S}$ maps services to conflict services.
- $\varepsilon : \mathscr{S} \rightarrow E$ maps services to the system resource usage sets (e.g. CPU, RAM, network transmission bandwidth usage etc.).

Example 4.1. *(Conflict Service)*

Let s_1 and s_2 be two services. Let $\mathscr{T}' \subseteq \mathscr{T}$. Assume s_1 has been requested by a real-time sensing request (e.g noise level sensing), which has its timestamp period within $\mathscr{T}'$, and s_2 is not requested by any client. Suppose $s_2 \in \kappa(s_1)$, then s_2 will be marked as unavailable during $\mathscr{T}'$.

4.2.4. Scalable Computational Resources

In Mist, it is expected that the SDM of a Mist node also describes the computational and networking capabilities (CPU, RAM, bandwidth etc.) it can provide. Since such information is available, a Mist node can form a grid computing group centred by itself with other Mist nodes that are within 1- hop range from it. Hence, when the Mist node cannot perform a task by itself or it cannot achieve the performance requirement for the task execution, it is possible to distribute the work (by executing a predefined substitute workflow pattern) to the other Mist nodes as long as it will generate a more efficient result. However, it raises a question about how does mePaaS makes the decision on which Mist node and when it should distribute the work to?

Here, the author proposes a work distribution scheme, which is used when a computational offloading node needs to be defined for the work substitution purpose at runtime.

Step 1. The resources required for executing the Work Item depend on the usage of the corresponding hardware components.

Let RES be the resource for the work item. RES consumes a set of hardware (CPU, memory, bandwidth, etc.; based on the historical record and input parameters). Let H be the hardware usage by RES, $H = \{h_k : 1 \leq k \leq \mathbb{N}\}$, where each $h \in H$ is defined as a tuple ($< id, value >$) corresponding to the identification of the hardware usage and the hardware usage value consumed for executing the RES.

Step 2. The weight of hardware usage required for the work item influences the performance ranking.

Based on the resource for the work item, the weight of hardware usage is different. It can categorise them into following types based on the hardware usage considered in [114] and [81]:

1. CPU intensive task (e.g. customised complex algorithm scripts).
2. CPU+RAM intensive task (e.g. I/O data processing; large data volume loading involved tasks)
3. Bandwidth intensive task (e.g. data forwarding process; e.g. send/receive tasks)
4. Hybrid, where multiple resources have higher weight.

Example 4.2.

If (b) is the classification of the Work Item, then for example, hardware parameters being considered are CPU, RAM, Bandwidth, then mark CPU = 2, RAM = 2, Bandwidth = 1, corresponding to CPU and RAM both plus one. Hence, the weight of CPU will be 2/5, RAM will be 2/5, Bandwidth will be 1/5.

Step 3. Identify availability of the system resources –Schedule Manager updates the service availability information when it progresses a new request. The service availability is influenced by two factors: service conflicts and system resource availability(i.e. CPU, RAM, network bandwidth, etc.).

Let $\mathscr{S}^{t_x}$ be a set of scheduled service executions at timestamp $t_x \in \mathscr{T}$, $\mathscr{S}^{t_x} = \{s^{t_x}_m | 1 \leq m \leq \mathbb{N}\}$. $\kappa(s^{t_x}_m)$ denotes a set of conflict services of $s^{t_x}_m$. Hence, a set of conflict services at timestamp—t_x (denoted by $\mathscr{K}^{t_x}$) will be:

$$\mathscr{K}^{t_x} = \bigcup_{m \in |\mathscr{S}^{t_x}|} \kappa(s^{t_x}_m) \tag{4.1}$$

and the available services at timestamp t_x (denoted by $\overline{\mathscr{S}^{t_x}}$) will be:

$$\overline{\mathscr{S}^{t_x}} = \frac{\mathscr{S}}{\mathscr{K}^{t_x}} \tag{4.2}$$

The above process has only filtered the services based on conflicts. Following is the process that considers the system resource availability. Let $E^{sys} = \{e_o^{sys} | 1 \leq o \leq \mathbb{N}\}$, be a set of the available system resources (Note that available system resources are different to the hardware specification of the mobile device. User can set the availability in percentage to avoid the service provisioning affecting to the normal use of the mobile device).

Let $\mathscr{S}^{t_x}$ be a set of services assigned at $t_x \in \mathscr{T}$. For each service—$s_z \in \mathscr{S}^{t_x}$, its system resource consumption is found in $\varepsilon(s_z)$. Let $E^{s_z} = \varepsilon(s_z)$ in which $E^{s_z} = \{e_o^{s_z} | 1 \leq o \leq \mathbb{N}\}$ in which the system resource denoted by $e_o^{s_z}$ and e_o^{sys} are the same, and let $ve_o^{s_z}$ be the usage value of $e_o^{s_z}$ and ve_o^{sys} be the remaining value of e_o^{sys}. For each t_x, the ve_o^{sys} after assigning $\mathscr{S}^{t_x}$ (denoted by $ve_o^{t_x}$) will be:

$$ve_o^{t_x} = ve_o^{sys} - \sum_{s_z \in \mathscr{S}^{t_x}} ve_o^{s_z} \tag{4.3}$$

Let $E^{t_x} = \{ve_o^{t_x}\}$. Referring to previous result, $\overline{\mathscr{S}^{t_x}}$ is a set of services that has been identified as available at t_x. For a service—$s_y \in \overline{\mathscr{S}^{t_x}}$, let E^{s_y} be the system resource usage required by the service. If $\exists e_o^{s_y} \in E^{s_y}$, such that $ve_o^{s_y} > ve_o^{t_x}$, which indicates that the service–s_y requires higher usage value than the actual available resource value. Hence, the s_y is considered as unavailable at the timestamp—t_x.

Step 4. The ranking of candidate – is based on the weight of resource and the resource availability.

Let $\mathscr{M}$ be a set of candidate Mist nodes where $\mathscr{M} = \{\mu_i : 1 \leq i \leq \mathbb{N}\}$. Each $\mu \in \mathscr{M}$ has a set of available hardware usage $\mathscr{A} = \{\alpha_j : 1 \leq j \leq \mathbb{N}\}$. Each α_j is defined as a tuple $< id, value >$, and $value_j^i$ denotes the value of available hardware usage α_j of Mist node μ_i. The score of a candidate Mist node—$\mu_x \in \mathscr{M}$ is computed by (4.4).

$$score_x = \sum_{j \in |\mathscr{A}|} \left(\frac{value_j^x}{\sum_{i \in |\mathscr{M}|} value_j^i} \times w'^x_j\right) \tag{4.4}$$

where w'^x_j denotes the normalised weight of the hardware usage α_j at μ_x, which is derived from (4.5).

$$w'^x_j = \frac{w_j^x}{\sum_{k \in |H^x|} w_k} \tag{4.5}$$

where w_j^x is the initial assigned weight value (see Example 1) and w_k is the weight of a $h_k \in H^x.H^x$ is the H^{Glo} of $\mu_x \in \mathscr{M}$.

4.3. Summary

In this chapter, the author has presented a mobile-embedded Platform as a Service (mePaaS) framework. The framework follows the service-oriented ESB architecture, which adapts the mechanism supported by the device (e.g., Wi-Fi communication, Bluetooth communication, GPS, etc.) into service. The author has extended the original work by adding service scheduling and the management of conflicts in order to improve the quality of service provisioning. With the combination of the service scheduling and script language supported BPMN workflow engine as the program execution engine, mePaaS allows the MWoT devices to provide a flexible platform for proximal users to offload their computational or networking program to mePaaS-based Mist Computing node. Details of the implemented prototype of mePaaS together with performance evaluation will be discussed in Chapter 6.3

5. ENERGY-EFFICIENT MOBILE WEB OF THINGS USING PUBLIC FOG NETWORKING SERVICES

5.1. Introduction

The fragmented interoperability issue of IoT [3] has motivated the incubation of the WoT [77], which applies Web technologies to IoT in order to enhance the integration among the IoT entities towards accelerating the development and the deployment of IoT ecosystem. Essentially, WoT systems are the IoT systems with additional mechanisms that enable the connected things such as home appliances, vehicles, animals and so forth, to be communicable via World Wide Web (i.e.. the Web) technologies, which belong to the application layer of OSI model. Implicitly, by utilising WoT, IoT developers can greatly reduce the fragmentation derived from the common practical implementation of IoT in which vendors are using their custom heterogeneous protocols.

While various domains have integrated IoT to their systems, MIoT emerged as the major theme in IoT that representing Internet connected moving objects such as human, animals, drones, vehicles and so forth, which often, are relying on battery-powered devices. Commonly, if all the MIoT system uses the devices from one single vendor, it may not have any issue in integrating the devices. However, as discussed earlier, such a statement reduces the flexibility of choosing the device vendor. Instead, if the system has applied WoT, in which the system will use only the WoT standard-compliant devices, it will increase the sustainability of the MIoT deployment and we term the WoT-driven MIoT as Mobile WoT.

Although MWoT is promising, it also raises numerous challenges in terms of processing power, storage space and the constraint energy because MWoT is commonly deployed on the battery-powered devices and the mobile Internet communication consume significant battery power. Accordingly [42], Wi-Fi network communication consumes much less energy than the cellular Internet. Ideally, If MWoT devices can collaborate with proximal Wi-Fi Internet access point, they can save significant battery power. Fortunately, recent research in IoT has emphasised the importance of distributing processes from the distant central server to the proximal resources such as programmable network routers, gateways, bridges or their co-located machines. Specifically, industry terms such a paradigm as Fog Computing and Networking architecture (i.e. the fog), which can cater the alternative Internet connect for MWoT.

It is foreseeable that in near future, the local small businesses and individuals will be providing public fog to the general public, which is similar to the Indie Fog business model and The Things Network. Further, the requester can utilise proximal fog servers on the move for the Internet communication instead of fully relying on the cellular mobile Internet.

Although utilising FogNets is a promising approach to improve the energy-efficiency of MWoT, it also raises a new research question: in an environment that consists of many heterogeneous FogNets providers that have different workload, queue and operation schedules, if the MWoT device randomly selects a FogNets provider simply based on the FogNet providers' availability, it may consume unnecessary energy for the MWoT device if the MWoT device switches the connected FogNets provider too frequent. On the other hand, if the MWoT device only switch FogNets provider after it loses connection or encounter issues with the FogNets provider, it will cause extra offline time period, which is further reduce the Quality of Service (QoS) of the MWoT and also can cause critical issues especially when the system utilises MWoT for remote healthcare.

In order to address the question, we propose a proactive FogNets scheduling framework for MWoT. Specifically, the proposed framework aims to optimise the FogNets connection schedule towards reducing the extra energy consumption derived from the switching FogNets providers.

5.2. System Design

A typical strategy to reduce energy consumption in this environment is to reduce the need of data transmission over the mobile Internet, because the mobile Internet consumes the most energy in the process. A common approach is to select one of the gateway devices in the proximity which is able to work as a broker for uploading the data to the corresponding organisations' Web server. As mentioned previously, if the environment has a pre-configured infrastructure, such a collaborative network has no issue. However, in IoT scenarios, the environment consists of numerous WSN groups that are operated by different organisations and it is not an easy task to enable the inter-organisational collaboration between those groups.

5.2.1. Overview

Figure 18 illustrates a federated environment where different organisations can perform collaboration in IoT data acquisition. The collaboration is based on utilising the Message Bus service managed by a trustworthy Coordinator server in the Cloud. The Message Bus routes the messages and data among participants. A participant can publish its services to the Message Bus and let the other participants subscribe to the services. Further, it enables the negotiation process between different participants towards assisting their collaboration.

In this environment, we made the following assumptions:

a) The movement of the MWoT device (D.1) is not very fast, roaming in a particular area and utilises standard proximal communication protocols for discovering its surrounding devices. Because of the lightweight proximal protocols (eg.BLE) , the energy consumption for discovering surrounding devices is very low compared to uploading data [41].
b) The MWoT devices periodically upload sensed data to their servers (Eg. in every 5 minutes).

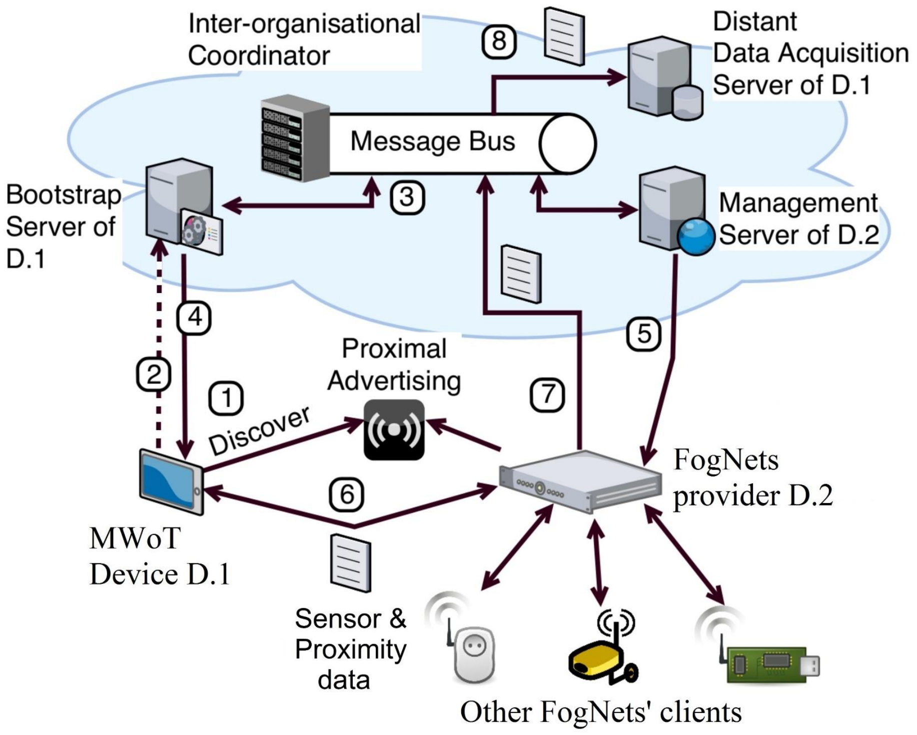


Figure 18: Fog Networking service example

c) The organisations those are willing work as FogNets providers for the other organisations, will use meaningful identification for their Fog nodes. Further, they publish the information (e.g. based on WSDL, WADL, CoAP etc.) of their Fog nodes in the Message Bus service together with the methods that help the other participants to establish the collaboration.

d) The Fog node is not always in active mode due to the energy conservation reason. They are discoverable via direct mobile P2P communication range while they are awake.

e) Inter-organizational collaboration involves privacy and service level agreement (SLA) concerns. For example, when organization A is willing to share its bandwidth with organization B, both organizations should come up with an agreement that consider the main facts about data transfer. For instance, data security, the quota of the shared bandwidth, how data will be treated at different nodes, how long it will take to deliver data to the destination, the maximum size of the data bundle, etc. will be considered as the main facts of SLAs. It is assumed that entering into SLAs is the responsibility of the management of the respective organizations. In addition, once organization B uploads data to organization A's Fog node, the data will be forwarded to organization B's servers that are under the control of organization A. Hence, this work, cannot consider how the data will be routed to a particular destination within the respective organization's network. The required SLAs are already being established.

In this example scenario the MWoT device is running a background operation which utilizes the low-powered proximity based service discovery procedures (e.g., browsing Bluetooth Advertisement, Wi-Fi Direct, WFi Aware, etc.) to locate devices that can possibly be the FogNets providers (mark 1). Then MWoT device sends the collection of the identification information of potential FogNets providers together with its current geographical location information to the Bootstrap Server (mark 2). (It is also worth to mention that at the very beginning the MWoT device utilises the mobile Internet connection to upload its proximity data, but subsequently it can use one of the selected FogNets provider to upload the proximity information.) Then Bootstrap Server (BS) can identify what are the suitable FogNets providers in D.1's current vicinity via the information retrieved from the Coordinator server. Afterwards, BS will request for collaboration from the management servers of the selected Fog Node (mark 3). Once the request is confirmed, BS will inform MWoT device about the confirmation with the information of the Fog Node (mark 4). Meanwhile, the management server of the selected FogNets provider (denoted by D.2) will configure D.2 to enable the communication from D.1 (mark 5). Afterwards, D.1 can use D.2 as its FogNets provider to upload data. The communication between the two devices can be in any short-range, low power wireless communication protocols depending on their configurations (mark 6). To preserve the energy of D.1, It can also upload other configuration data (eg. identification information of potential FogNets providers) over the same connection instead of utilizing mobile Internet that will consume more energy. As the figure shows, initially the D.2 is work as a gateway of the other FogNets clients to send data packages to the Message Bus. Since D.2 may send all the data it collects (for different parties) in one package to the Message Bus (mark 7), the Message Bus will decompose the data collection and forward to the corresponding channels. The participants who subscribed to the corresponding channels will automatically receive the data. In this scenario, the Coordinator would have approved the Distant Data Acquisition Server (DDAS) of D.1 for subscribing the data derived from D.1. Hence, the DDAS of D.1 can properly receive the data collected by D.1 (mark 8).

5.2.2. Main components of the proposed framework

The environment may consist of four main elements.

Other FogNets' clients

Other FogNets clients' devices are classic sensor devices that are deployed in the environment to collect specific data (e.g. temperature, noise level, humidity, etc.). Those are often low powered devices with limited application logic and some of them may be working as a sink node for the certain amount of sensor nodes.

Mobile Web of Things (MWoT) device
Mobile WoT device is a battery-powered device that has more processing and energy capabilities for sensing the environment on the move. It can be carried or attached to any moving objects (human actors, animals, robots etc.). The device can have inbuilt sensors and also capable of collecting data from proximity sensors. Initially, MWoT device uploads its collected sensory data directly to its Distant Data Acquisition Server (DDAS) via the mobile Internet connection. However, the frequent data upload via mobile Internet will drain the battery quickly. Hence, utilising the proximal Internet-connected devices to forward data to its DDAS will help MWoT device to preserve energy for longer life.

FogNets provider device D.2
We consider that FogNets provider devices are work as potential gateways for the MWoT devices. Normally they may have a better energy source and more computational/ storage resources than the MWoT Devices. D.2 will collect data from the own FogNets client devices, and upload the collected sensor data to their organization's DDAS. Based on the organization's preferences, FogNets provider devices are discoverable by using common standard proximal communication such as Bluetooth, Wi-Fi direct, etc. In this environment, we assume that a discoverable FogNets provider devices will advertise a URL of its management server for further information (eg. communication protocols, availability, security credentials, etc.) for collaborative IoT data acquisition. Also, please note that FogNets provider devices may not be available for continuous data acquisition due to some limitations such as technical failures, organization policies, energy/bandwidth limitations, etc.

Distant Data Acquisition Server
Each organization may have a separate server to acquire data coming from its own sensor nodes. Yet, the way of delivering data from sensor nodes to the DDAS may be in different ways. For example, in the beginning, the collected data from the sensor nodes could be directly forwarded to the respective DDAS by MWoT device and later the data may be received by the DDAS through a different organization's network.

Bootstrap Server
BS directly communicates with the MWoT device to collect information about potential FogNets providers in proximity. Thereafter, BS will request for collaboration from the management servers of the selected FogNets and prepare the optimal schedule for the corresponding sensing period that minimizes the number of re-connections.

Inter Organizational Coordinator
This entity works as a top-level coordinator for all organizations. All organizations willing to collaborate will exchange messages and data over the Message Bus service which provides a communication path across all entities. The coordinating server processes the requests for collaboration from different organizations shared with other organizations over the Message Bus.

Backend Management Server of Organisation (MS)

MS is the controller of an organisations Fog nodes and it can control one or many Fog nodes and MWoT devices too. The MS has the following main responsibilities:

a) Providing SDM of a Fog node. As mentioned previously, a Fog node can reply with the URL of the MS controlling it. If a requester retrieves a URL from the Fog node, it then performs a HTTP GET request on the MS, and the MS will reply with a SDM that describes what types of operations are available from the MS. For example, the SDM can describe what data the corresponding Fog node has collected and the URI to retrieve the data from corresponding DAAS. Also, the SDM describes how the collaborative network can be established. The details of the SDM will be described in a later section.

b) Configuring the processes of Fog nodes. Each time the MS receives data from the Fog node by the classic request/response method, the MS will check if the configuration of Fog node needs to be updated. If so, the MS will wrap the updated configuration setting metadata with the response message and send it to the Fog node. The Fog node will then check the response message. If re-configuration is required, it will perform the re-configuration. For example, the reconfiguration may include a request to Fog node to collect data from new sensor nodes that have been deployed in the environment by the organisation.

c) Establishing and negotiating the collaboration with other organisations. Fundamentally, each organisation manages its own WSN, and their Fog nodes/ MWoT devices which are uploading data individually. For example, in Figure 18, initially, the MWoT device is uploading data to DDAS directly over the mobile Internet that consumes more energy. However, a common approach is to select one of the FogNets provider in the proximity which can work as a broker to upload the data.

5.2.3. FogNets provider scheduling

As mentioned earlier, we consider that FogNets providers may have different availability due to some circumstances. Switch between many FogNets providers consumes more energy than connected to the same provider during the data uploading cycle of the MWoT device. Hence, it is necessary to provide the scheduling scheme to improve the energy efficiency of the MWoT device when selecting a FogNets provider and establish collaboration. Accordingly, the BS prepares the best schedule for the corresponding MWoT device that minimise the number of switching between the FogNets providers.

We formalise the scheduling scheme as below:

Let $\mathscr{T} = \{\tau_i : 1 \leq i \leq \mathbb{N}\}$ be a set of discontinuous task periods assigned from the BS to the MWoT device by following the time flow order. e.g. τ_1 denotes

3:00∼3:05, τ_2 denotes 3:15∼3:20, and the time represented by τ_2 must not be prior than τ_1.

Let $\mathbb{C} = \{C_i : 1 \leq i \leq \mathbb{N}\}$ be a set of candidate FogNets providers for the task periods. Where C_i denotes a set of candidate FogNets providers for task period—$\tau_i \in \mathscr{T}$.

Let $H = \{h_j : 1 \leq j \leq \mathbb{N}\}$, be a set of all the FogNets providers in proximity. To match the available time of h_j to each $C_i \in \mathbb{C}$, we use Algorithm 5.1.

Algorithm 5.1 Matching with the schedules of FogNets

```
 1: for $\tau_i \in \mathscr{T}$ do
 2:     $ST_i \leftarrow$ the start time of $\tau_i$
 3:     $ET_i \leftarrow$ the end time of $\tau_i$
 4:     $C_i \leftarrow$ new set
 5:     for $h_j \in H$ do
 6:         $T_j \leftarrow$ set of scheduled online time of $h_j$
 7:         for $t_k \in T_j$ do
 8:             $ST_k \leftarrow$ the start time of $t_k$
 9:             $ET_k \leftarrow$ the end time of $t_k$
10:             if $(ST_k < (ST_i + \texttt{buff}_{ST_i})) \wedge (ET_k > (ET_i + \texttt{buff}_{ET_i}))$ then
11:                 add $ID$ of $h_j$ to $C_i$
12:             end if
13:         end for
14:     end for
15: end for
```

After applying Algorithm 5.1, we obtained a set of candidates for each $C_i \in \mathbb{C}$. The buff times mentioned in the algorithm are the buff time for start time ($\texttt{buff}_{ST_i}$), which is the discovery and connection time, and the buff time for the end time ($\texttt{buff}_{ET_i}$), which is the time required to complete the last data transmission before disconnection.

Here, we consider that re-connection will consume extra power. If the energy consumption of re-connection is higher than maintaining the same connection, then maintaining the same connection between two scheduled periods is more preferable. In order to find out if there is any possibility to use the same FogNets provider for two sequential ordered periods, we first need to identify the sequential ordered candidates in $\mathbb{C}$.

Let a_j and b_j be the representation of h_j assigned in two sequential schedule time groups. The h_j will be added to the sequential schedule list—$\mathscr{P}$ when $\mathscr{P} = \{(a_j, b_j) : \exists a_j \in C_{i-1}^{b_j}\}$, where $C_{i-1}^{b_j}$ denotes the previous time group of the b_j's time group.

After $\mathscr{P}$ is generated, we can use the information provided by $\mathscr{P}$ to rank each element—$c \in C_i$. i.e. c_x denotes the ID of h_x. Suppose c_x is one of the element in

C_i, to compute the ranking of c_x, we use 5.1.

$$rank(c_x) = \begin{cases} 1 + \sum_{m=0}^{\eta} |\{\exists c \in C_{i+m}^{c_x} : c = c_x\}| & \texttt{if} \text{ has already connected.} \\ \sum_{m=0}^{\eta} |\{\exists c \in C_{i+m}^{c_x} : c = c_x\}| & \text{otherwise.} \end{cases} \tag{5.1}$$

where η denotes the set index number where the node represented by c_x is no longer found. $C_i^{c_x}$ denotes the schedule time group of c_x. $C_{i+m}^{c_x}$ is the next schedule time group of $C_i^{c_x}$.

The already connected FogNets device will be considered as priority. Hence, the ranking value is added by 1.

Afterwards, in a C_i, if two or more candidate FogNets providers have same ranking value, the connection will be retained. Otherwise, the highest ranked node will be selected. If currently there is no connection and there are multiple same ranked nodes, the MWoT device will randomly select one of them.

Originally, when two or more options have same ranking value, then further ranking is made based on other context attributes. However, here we do not include the other attributes that influence the ranking of the node. The corresponding scheme has been discussed in the previous work [25]. We plan to compose them in our future work.

5.3. Summary

In this chapter, the author proposed a framework for energy-efficient mobile data acquisition using opportunistic FogNets devices. In this environment, the inter-organisational FogNets devices can act as gateway service providers in order to collaboratively conserving energy usage from the mobile Internet-based data transmission. Further, the author considers the dynamic nature of FogNets devices in terms of their scheduled availability. Hence, the author introduced a proactive gateway service scheduling scheme to facilitate the establishment of the collaboration. The scheduling scheme aims to reduce the re-connection between the collaborative devices in order to minimise the unnecessary energy consumption derived from re-connection processes. The proof-of-concept prototype evaluation will be discussed in Chapter 6.4

6. PROTOTYPE IMPLEMENTATION AND EVALUATION

6.1. Introduction

In the previous chapters, the author has described the proposed frameworks to enable energy efficient MWoT service provisioning. Particularly, in Chapter 3, the author has presented the architecture of the energy efficient, lightweight MWoT service provisioning framework. Next, in Chapter 4, the author has described a service-scheduling feature that addresses the challenges when processing a large number of concurrent client requests in the constrained environment. In addition, in Chapter 5 introduced a proactive gateway service scheduling scheme in the opportunistic Internet sharing environment that optimises the energy consumption of the MWoT devices.

In order to validate the approaches discussed in the previous chapters and for proof-of-concept, the author has developed and evaluated a number of prototypes and use cases. This Chapter presented the evaluation methods and results of our prototypes.

6.2. Evaluation of Energy-Efficient Mobile Web of Things using Public Fog Networking Services

6.2.1. Objective

As was discussed in Chapter 5, MWoT devices can reduce mobile Internet usage by using network services from proximal FogNets providers. A scenario where the MWoT device works as a data collector for an organization and utilizes other organizations has been presented. Fog gateways in proximity upload the collected data without using services of the mobile Internet. Further, an optimal schedule to reduce switching between Fog gateways when there are many FogNets devices in proximity, has also been presented. To evaluate the performance of the proposed framework, a prototype has been implemented to conduct several test cases, which have already been done. The following section provides details of test scenarios and the results of performance evaluation.

6.2.2. Experimental Setup and Prototype Implementation

The evaluation has been performed using LG Sprit and G4 mobile phones running Android version 5.0.1. The FogNets provider device has embedded with NanoHTTPd server (`https://github.com/NanoHttpd`). The mobile Internet used by the devices is TELE2 4G mobile Internet connection. To measure the energy consumption, our test bed consists of PeakTech Digital Multimeter which provides the visualized real-time energy consumption logging of the mobile devices (Figure 19). The Multimeter is coupled to the battery of the MWoT device and measures the current flow and the voltage level during the experiment.

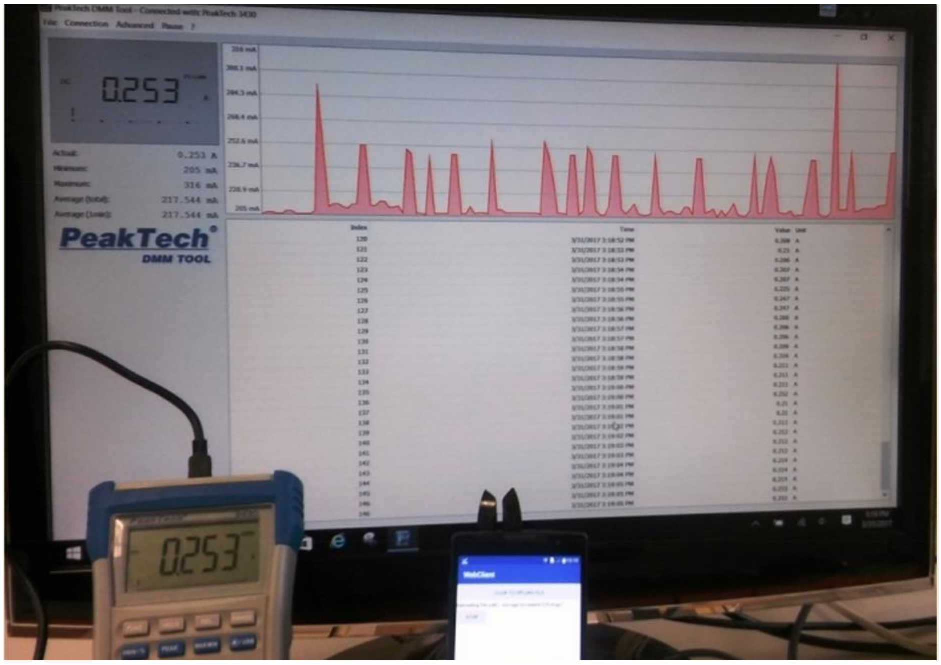


Figure 19: Testbed for the energy measurement

Case Scenario 6.2.1. ***Power Consumption due to Switching Wi-Fi Networks***

First, we designed a test scenario to demonstrate the power consumption pattern of a mobile device when it switches from one Wi-Fi network to another. Here, the mobile device is connected to a Wi-Fi network and we obtained the power consumption trace for 200 seconds duration. As shown in Figure 20, always connected to the same Wi-Fi gateway for 200 seconds consumes about 136.45 Joules. Next, we switched the Wi-Fi connection frequently during the same time period and obtained the power traces accordingly. It is clearly shown that switching between Wi-Fi gateways consumes a considerable amount of energy than maintaining the same connection. For example, it consumes additional 39 Joules (175.6-136.45) when switching between five gateways instead of maintaining the same gateway.

Case Scenario 6.2.2. ***Optimised FogNets provider Scheduling***

This test scenario presents the experimental results of the proposed scheduling algorithm. In the setting of this scenario, the MWoT device has been scheduled 5 periodical tasks which of each require 3 minutes sensory data uploading to the DDAS through a FogNets provider. The scheduled start time has five time stamps as at 4:00, 4:05, 4:10, 4:15, 4:20 and 2 minutes buffer time also allocated for both start time and end time. In the experimental situation we have 7 FogNets providers

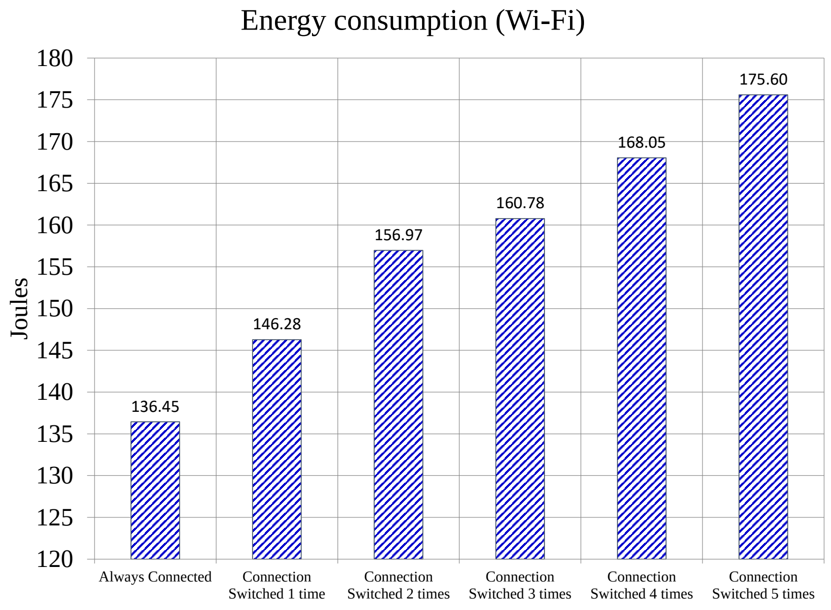


Figure 20: Energy consumption due to switching Wi-Fi Networks

as A, B, C, D, E, F and G and the available connection period as follows: A 3.50-4.10 and 4.30-4.50, B 4.05-4.25 and 4.45-5.05, C 3.40-4.10 and 4.15-4.45, D 3.15-3.40 and 4.10-4.30, E 3.45-4.15, F 3.35-3.50 and 4.05-4.30, G 3.55-4.10 and 4.20-4.40. In the first scenario the MWoT device selects the FogNets providers for each time period randomly (without considering the scheduling algorithm) as shown in Table 3a. In this test case we assume that the QoS of all the candidate FogNets providers' is similar. Next time, the MWoT device selected the best FogNets provider based on the optimised schedule (Table 3b) that intends to minimize the need of switching FogNets providers during the data uploading phase. During the experiment we measured the power consumption of the MWoT device according to the random schedule and the optimised schedule.

At the beginning, the MWoT device that uses the random schedule, does not connect to any FogNets provider, but it keeps scanning for available providers. Based on the random schedule shown in Table 3a, at the time stamp T1 it established a connection with the FogNets provider-B and at the time stamp T2 the MWoT device disconnected from the FogNets provider-B and connected to the FogNets provider-E. Since the schedule is random, the MWoT device disconnects and connects to available FogNets providers for three times (C –> E –> B –> C). As shown in Figure 21, the re-connection processes cause the device to consume more power than with maintaining a stable connection.

On the other hand, in optimised scheduling approach, the MWoT device initially connected to the FogNets provider-E at the first time stamp and maintains the same connection until the time stamp T3. At the time stamp T3 it switches to

Table 3: Scheduling cases; X denotes the availability, Ⓑ denotes the buff time; ⊕ denotes the selected schedule.

(a) Random schedule. Switch FogNets provider three times

Time Slot	A	B	C	D	E	F	G
T1 - 4:00-4:05	X		⊕		X		X
T2 - 4:05-4:10	Ⓑ	Ⓑ	Ⓑ		⊕	Ⓑ	Ⓑ
T3 - 4:10-4:15		⊕		Ⓑ	Ⓑ	X	
T4 - 4:15-4:20		⊕	Ⓑ	X		X	
T5 - 4:20-4:25		Ⓑ	⊕	X		X	Ⓑ

(b) Optimised schedule. Switch FogNets provider one time

Time Slot	A	B	C	D	E	F	G
T1 - 4:00-4:05	X		X		⊕		X
T2 - 4:05-4:10	Ⓑ	Ⓑ	Ⓑ		⊕	Ⓑ	Ⓑ
T3 - 4:10-4:15		X		Ⓑ	Ⓑ	⊕	
T4 - 4:15-4:20		X	Ⓑ	X		⊕	
T5 - 4:20-4:25		Ⓑ	X	X		⊕	Ⓑ

the FogNets provider-F and maintain the same connection for the whole scheduled period. Based on the optimised schedule which is shown in Table 3b the MWoT device switched its gateway only one time. Hence, it consumes less power than the random selection approach.

Case Scenario 6.2.3. ***Energy consumption of the Mobile WoT device***

In this test case we monitored the average energy consumption of the mobile devices when it is uploading data to its Distant Data Acquisition Server (DDAS) in different ways.

Self-upload

Here, we measured the energy consumption at the MWoT device when uploading data to its own DDAS over 4G mobile Internet. The dataset consists of the items that are of size 200KB, 400KB, 600KB, 800KB and 1000KB respectively. Measured average energy consumption to upload each item is varied (as shown in Figure 25) from 267.9 Joules (for 200KB) to 296.7 Joules (for 1000KB).

Via a FogNets provider

In this scenario, the MWoT device uploads data to the DDAS through another mobile device that works as a FogNets provider. To measure the energy consumption of the MWoT device, we designed a test case that follows the steps mentioned in Figure 15.

The first step is measuring the energy consumption for proximity scanning and uploads the retrieved data to the BS over the mobile Internet. Hence, we

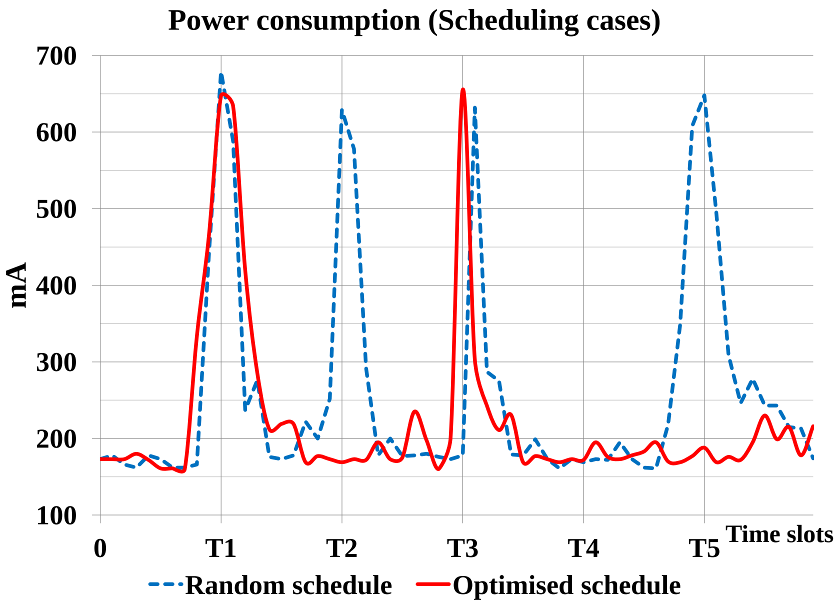


Figure 21: Power consumption (optimised schedule Vs. random schedule)

configure five FogNets provider devices (mobile phones) that advertise their management servers' URLs as BLE advertisements. Also, we configured a BS server on the Heroku cloud Platform and an Amazon EC2 instance as a management server. Next, the MWoT device does the BLE scanning for 15 seconds, upload the collected information to the BS over the mobile Internet. At the BS server, it requested the relevant configuration information of the FogNets provider from the management server and send the optimized schedule to the MWoT device (in this experiment, it is just the name of the access point that associated with the FogNets provider device and the time slot). The energy consumption of the MWoT device for this task is varying from 29.59 Joules to 36.27 Joules due to the communication/processing delays at the servers. The second task is uploading the data to the selected FogNets provider device over the Wi-Fi. The MWoT device search for available Wi-Fi access points, established a session with the preferred access point (FogNets provider), and uploads 200KB of sensor data. Total energy consumption for this task is about 223.75 Joules. Figure 22 presented the energy utilisation of the MWoT device that follows the proposed framework in different sensing schedules in contrast to doing the solo sensing. The result has shown that when the numbers of sensing tasks are increasing the energy efficiency of the proposed framework also increasing.

Also, we have tested energy consumption of BLE scanning in different situations. For instance, Figure 23 shows that BLE scanning power trace of the MWoT device that is in the environment of minimum 5, 10, and 15 devices which are advertising their URLs. The average power consumption for these three situations

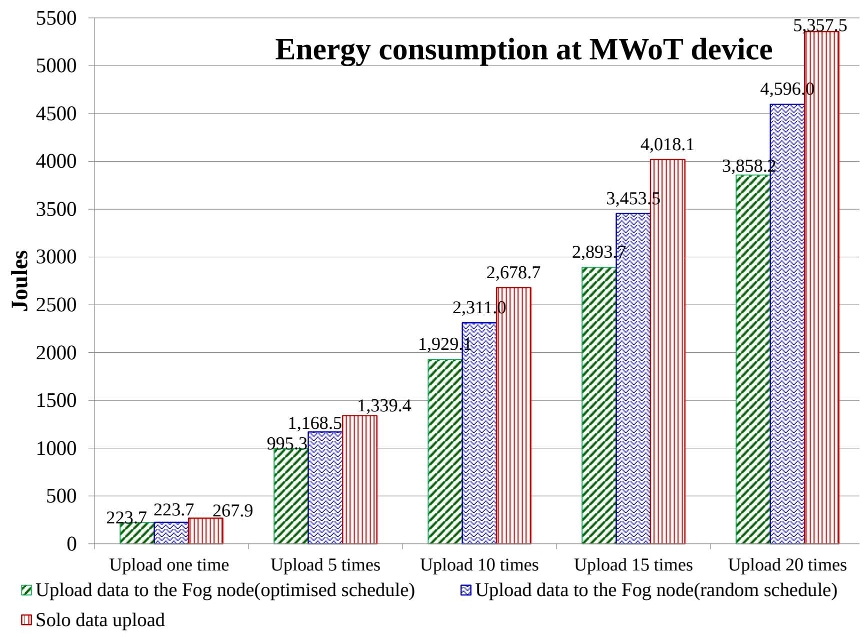


Figure 22: Energy utilisation of the MWoT device (doing solo sensing Vs. collaborative sensing)

is 219.96mA, 219.07mA, and 220.27mA respectively.

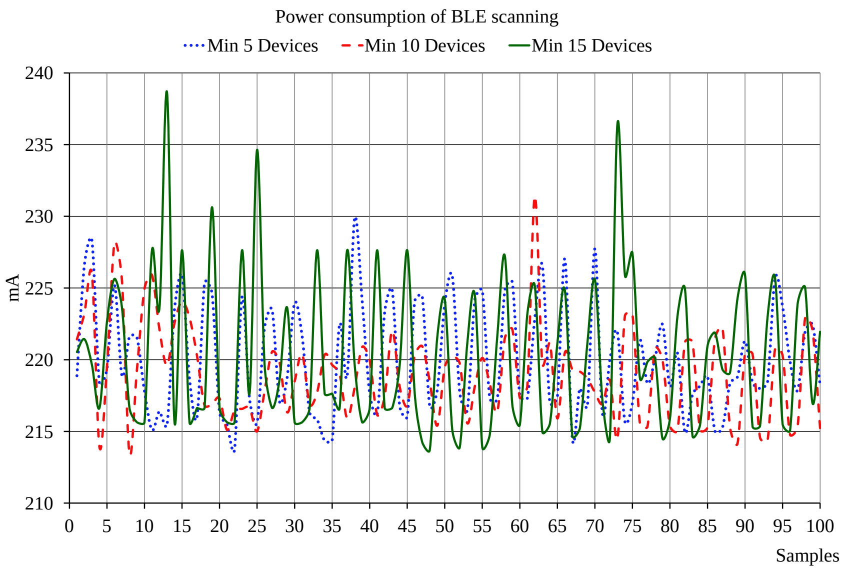


Figure 23: Power consumption for BLE scanning in three different environments

Apart from that, we also studied the average power consumption of the BLE scanning in the environment that has more than 15 BLE devices and for the period of 20, 40, and 60 seconds. As shown in Figure 24 there is a big power demand at the beginning of the scanning (about 400mA) and maintain the low power consumption (225mA) during the rest of time.

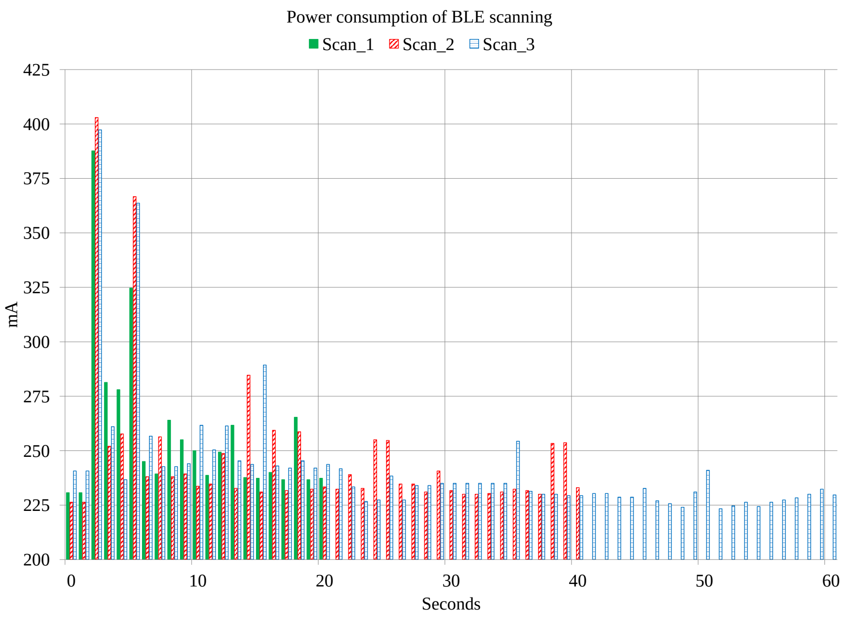


Figure 24: Power consumption of the MWoT device for BLE scanning(20, 40, 60 seconds)

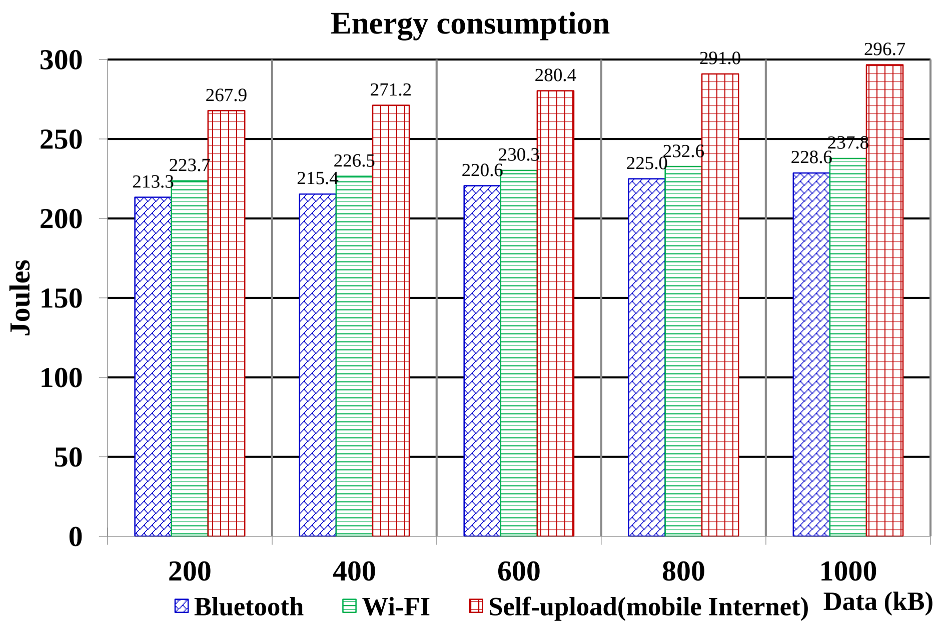


Figure 25: Energy consumption (Solo Vs. via a FogNets provider)

Moreover, we also conducted a test with two different proximity communication types and analysed the average energy consumption at the MWoT device as follows.

a) Upload the data over the Bluetooth channel
   In this case, MWoT device uploads its data to the FogNets provider device over the Bluetooth channel which is established between them. The size of uploaded data is the same as the previous setting (200KB-1000KB) and the results are shown in Figure 25.
b) Upload the data over the Wi-Fi channel
   Here, the FogNets provider device also works as a Wi-Fi hotspot and the MWoT device connected to the FogNets provider over the Wi-Fi link and upload the data to the Fog node (200KB-1000KB). The FogNets provider then forwards data to the corresponding DDAS at a later time. The average energy consumption of each instance is also presented in Figure 25.

Case Scenario 6.2.4. ***Being a FogNets provider Vs. not being a FogNets provider***

In this experiment we discovered the energy expenditure at the FogNets provider device when:

a) Uploads its own data collected from the sensors to the own DDAS
b) Also, working as a FogNets provider to upload other MWoT devices' data while uploading the own data

Moreover, the test was conducted with the same Sensor data set that consists of item's size as 200, 400, 600, 800, 1000KB. In the first instance, we assumed that the initial data size is 200KB and the FogNets provider uploads this data to the cloud server at regular intervals. The measured average energy expenditure for this transaction is about 269.1 Joules. In the second instance, the MWoT device sends the different size of data items (from 200KB to 1000KB) to the FogNets provider device and it uploads received data together with its own data (200KB) as one package. Figure 26 shows that the average energy consumption to upload only the own data(not being a FogNets provider) and to receive+upload the brokering data with own data (being a FogNets provider).

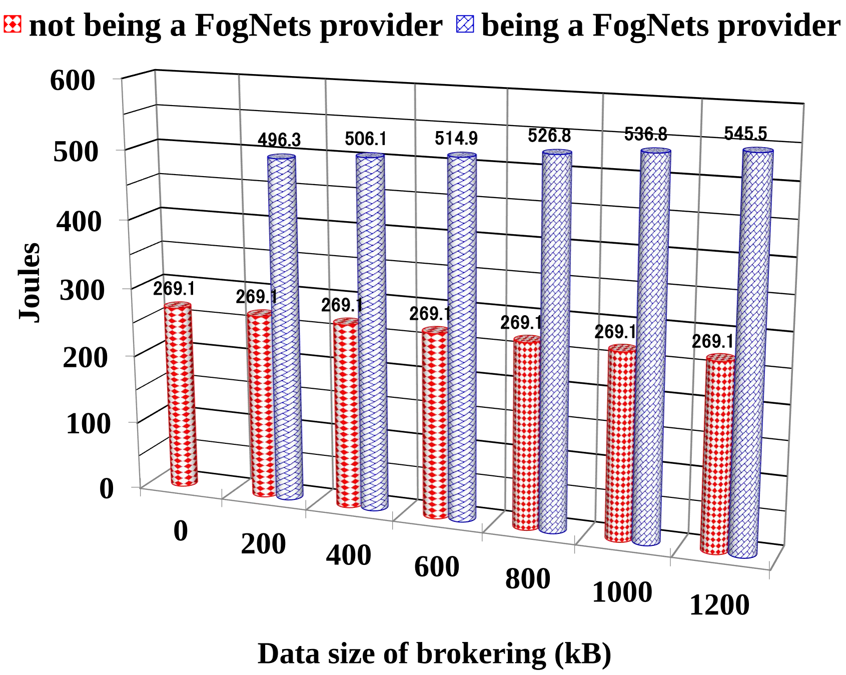


Figure 26: Comparison of energy consumption (default data size is 200 KB)

6.2.3. Discussion

We observed that direct uploading data over the mobile Internet connection is consuming more energy than using a FogNets provider. However, according to Figure 25, to upload 1000KB directly to the DDAS cost 296.7 Joules while uploading the same data to the FogNets provider through the proximity communication methods like Bluetooth and Wi-Fi, it cost only 228.6 Joules and 237.8 Joules respectively.

When a device works as a FogNets provider for other devices, we can see that there is no big difference in energy consumption when uploading the data within a certain limit. For instance, when the FogNets provider device is not brokering, it consumes 269.1 Joules to upload 200KB and while being as a FogNets provider for another 200KB from the MWoT device, it consumes only additional 227.2 Joules (496.3-269.1)(Figure 26). In summary, when comparing the energy consumption of being a FogNets provider vs. not being a FogNets provider, it is clear to see that being a FogNets provider consumes a little bit of more energy than the solo mode because being a FogNets provider involves more processing tasks (such as maintaining sessions, running a server, running hotspot, etc.) compared to the case of not being a FogNets provider. Apart from that working as a FogNets provider for the proximity devices is not a huge burden for the device.

6.3. Evaluation of Adaptive Mobile Web Server Framework for Mist Computing in the Internet of Things

6.3.1. Objective

The proposed mobile-embedded Platform, as a Service (mePaaS) framework (Chapter 4), enables mobile devices to provide a flexible way of sharing their computational and networking mechanisms as services. In the Mist Computing environment, the mePass framework provides a flexible program execution environment and self-adaptive resource management in mobile devices. Further, if a mePass node cannot perform a task by itself or if it cannot achieve the performance requirement for the task execution, it is possible to distribute the work to another mePass node as long as it generates a more efficient result.The performance of the proposed mePass framework has been thoroughly evaluated using several test cases. The objective of these test scenarios is to show the self-adaptive resource management and task distribution features of the mePass node that enhance the QoS of service provisioning in the Mist computing environment.

6.3.2. Experimental Setup and Prototype Implementation

The proof-of-concept mePaaS prototype has been implemented on an Android OS mobile device (LG G4C). Following is the summary of the main components that have been implemented for the prototype:

Controller, Local Service Module Manager (LSMM), Service Provisioning and Schedule Manager:. These components are the core elements of mePaaS. They have been implemented as one local service on Android OS.

Service modules were implemented as independent local applications. They are managed by LSMM component in which LSMM dynamically launches them as plugin based services for fulfilling the requests and they are automatically terminated when they are no longer needed. Since the service modules are a plugin-like software components, it is easy to extend the framework by installing more modules (Android application package). The current version of the prototype has seven service modules: Video, Image, GPS, Temperature, HTTP, MQTT, and CoAP.

Program Execution Engine is an extension of the Android-ported Activity BPM engine (http://activiti.org) derived from [44]. It can execute the program that has been modelled as BPMN with script language support. The details and performance testing of the process engine can be found in [44]. Although the ported Activity BPM may not be the best option for other kind of IoT devices (e.g. Raspberry Pi), at this stage, it is sufficient for the proof-of-concept of mePaaS.

Case Scenario 6.3.1. ***Self-configured service description of SDM***

This test scenario evaluates the feature of dynamic changes in the SDM according to available resources of the mePass node. First, the author has developed several service modules and installed with the mePaaS. Initially, in the idle state, the MWoT device is not processing any request and Local Service Module Manager has published the SDM as below:

```
{
"@context": "http://schema.org/geo","@id": "i:locaSensor1",
"@type": "i:LocationSensor",
"name": "Current Location of the Server",
"url": "http://172.19.28.237:8765/location",
"@context": "http://schema.org/temp",
"@id":"i:tempSensor1","@type":"i:TemperatureSensor",
"name":"Current Ambient Temperature",
"url":"http://172.19.28.237:8765/temp",
"@context": "http://schema.org/image","@id": "i:camera1",
"@type": "i:ImageSensor","name": " Image Sensing Service ",
"url":"http://172.19.28.237:8765/image",
"@context": "http://schema.org/video","@id": "i:camera1",
"@type": "i:ImageSensor","name": " Video Sensing Service ",
"url":"http://172.19.28.237:8765/video",
"@context": "http://schema.org/upload","@id": "i:upload",
"@type": "i:DataUpload1","name": "Data Uploading Service",
"url":"http://172.19.28.237:8765/dataForwader"
}
```

Since there are no resource intensive services are currently running, mePaaS published all service modules are available to clients. To confirm the dynamic nature of the service provisioning, then send a client request to the mePaaS node, that require a periodic video sensing task which captures a 30 second video clip in every minute for 10 minutes duration and then uploads to a distant server. This task requires using the camera as a hardware component and considerable amount of mobile Internet bandwidth. Moreover, the controller coordinates with the LSMM, Schedule Manager, and SP in order to update the SDM. In our previous prototype, the controller just removes the other services from the SDM those also require the camera to provide the services. However, with the information from the schedule manager, this new prototype can reconfigure the SDM more effectively. Because the controller provides exact time frames when the camera hardware is available in advance, hence the LSMM can inform to the SP to reconfigure the SDM accordingly. Moreover, in our framework, the SP component temporally removed two services from the SDM that utilised the camera hardware as shown below, to avoid receiving another service request that also needs to access the camera at the same time slot.

```
{
"@context": "http://schema.org/geo","@id": "i:locaSensor1",
"@type": "i:LocationSensor",
"name": "Current Location of the Server",
"url": "http://172.19.28.237:8765/location",
"@context": "http://schema.org/temp","@id":"i:tempSensor1",
"@type":"i:TemperatureSensor",
"name":"Current Ambient Temperature",
"url":"http://172.19.28.237:8765/temp",
"@context": "http://schema.org/upload","@id": "i:upload",
"@type": "i:DataUpload1","name": "Data Uploading Service",
"url":"http://172.19.28.237:8765/dataForwader"
}
```

However, this is only for 30 seconds and thereafter the SP component reconfigures SDM that will make available video and image sensing until the next time slot.

Case Scenario 6.3.2. ***Communication delay - Mist vs. Fog vs. Distant Data Centre***

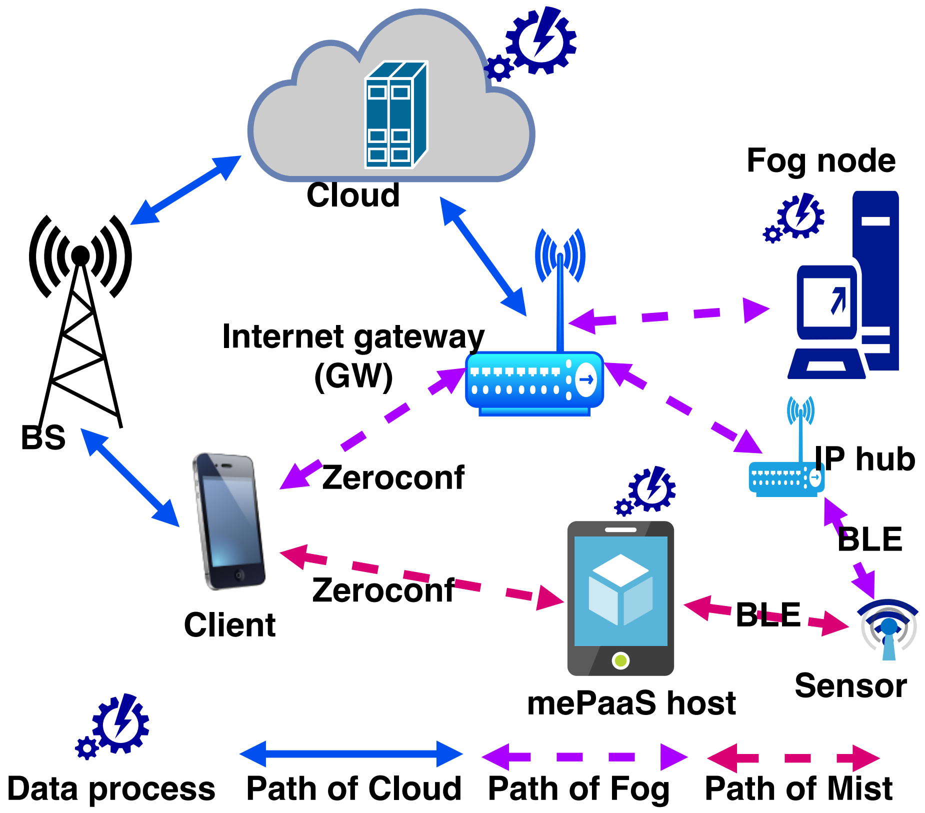


Figure 27: Experiment setup. Mist vs. Fog vs. Distant Data Centre

In this test case, the author compared the communication delay between three paths as cloud, fog, and mist. As shown in Figure 27, the author deployed a temperature sensor which connected to the Arduino board. The sensor advertises ambient temperature with the current time stamp in every second via the BLE communication protocol. In the first test case, mePaaS host collects temperature data in the proximity, (just from the sensor via BLE) and provide to other clients as a service. Moreover, the client requests the current temperature information from the mePaaS host and it just forwards the temperature information, including the timestamp which was recorded at the sensor node. At the client side, calculate the average time difference between the original and received timestamp for one thousand transactions. Secondly, the author measure the communication delay along the fog path. In this case, since the temperature sensor uses BLE as a local communication protocol, we need a proxy device that connects BLE to IP network. Here, we developed a simple Bluetooth application and installed it on a mobile phone that receives data over BLE and forwards to the IP network over the Wi-Fi link. Thereafter, the fog node received the temperature from the proxy device and makes it available to the clients within the local Wi-Fi network. Here, the fog node is a laptop (HP EliteBook G3) which connected to a TP-Link (TL-WR940N) Wi-Fi router which set up a local network. In this test case, the client connected to the local Wi-Fi network and received the ambient temperature over the established fog path. The author measured the average delay as same as that mentioned earlier. Finally, in the cloud path, the temperature sensor sends data to the proxy device, and then it moves to the cloud server over the local Wi-Fi network. Moreover, a Heroku web app has been developed as the cloud server, which receives temperature data from the local sensor and published as a service to other clients. In this situation, the client connected to the cloud server via a mobile Internet connection (TELE2 LTE connection-Estonia) with the average of 50 Mbps and 33.87 Mbps for download and upload speed respectively. As of other test cases, the author measured the average time delay for the cloud path also. Since the amount of data is very small please note that we neglected the processing time at the respective node in all test cases.

According to Figure 28, the mist path provides the least delay time (97.75 ms) for client requests while the cloud path has highest communication delay (1287.28 ms). Moreover, the fog path has the 719.08 ms delay because the sensor data travel through the proxy device and the fog server too.

Case Scenario 6.3.3. ***Performance of dynamic service module execution***

As mentioned earlier, since the mePaaS node launches service modules on demand, it has performed the bootstrapping test for each implemented service module. Moreover, the author concerned that the bootstrapping process of service modules can influence the overall performance and also add extra cost (e.g. energy, which is important if the Mist node is running in uncharged mode). Hence, in this test case, the author measured the latency and the energy consumption that are caused by bootstrapping the mePaaS service modules (Figure 29).

First, the author performed performed a test to get the average boot-up time for each module. According to Figure 29(a), MQTT and CoAP service modules have explicit latency due to establishing the underlying protocol stack. Also, here the HTTP server is based on AndroidAsync
(`https://github.com/koush/AndroidAsync`) that provide lightweight web server running on Android. However, we can see a little bit higher delay in the Temperature

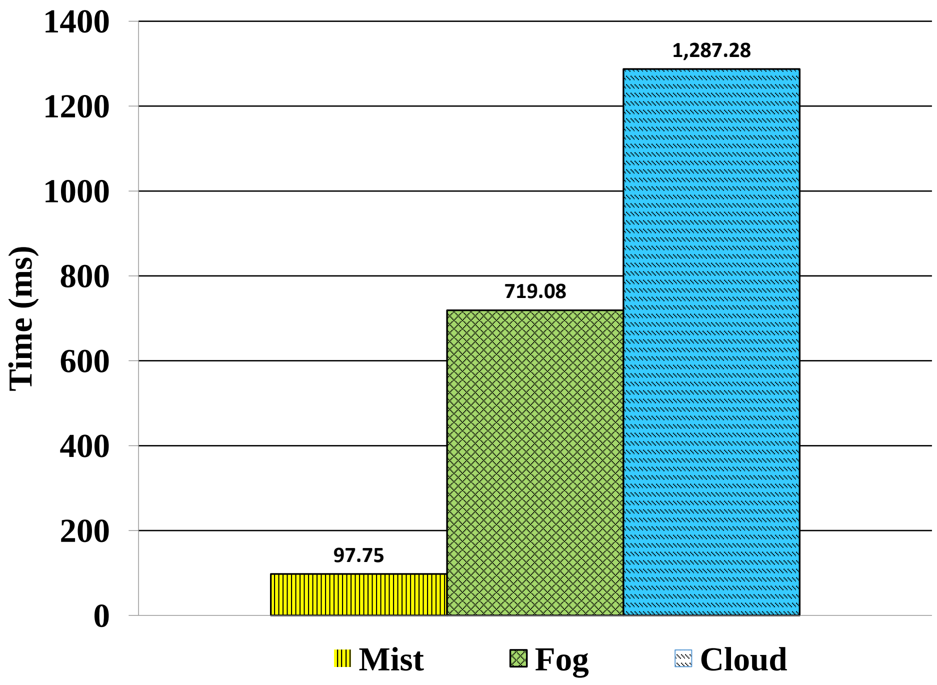


Figure 28: Communication latency

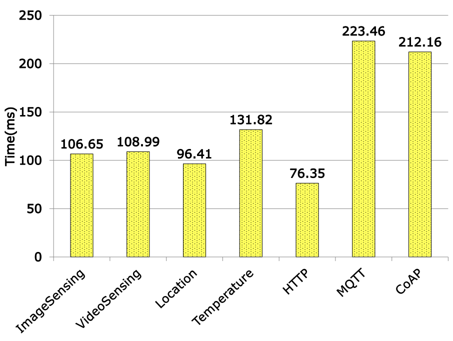
sensing module. The reason is that there is no inbuilt temperature sensor; the module should fetch the data from the proximity sensor. First, the module should establish a BLE connection with the proximity sensor device that caused a few milliseconds delay.

Second, the author performed a test to get the average power consumption of modules during the bootstrapping. According to the test results (Figure 29(b)) that measured by PeakTeck 3430 Digital Multimeter, the MQTT module consumes the highest power (212.2mA) while CoAP and the Temperature sensing modules also consume a reasonable amount of power during the bootstrapping.

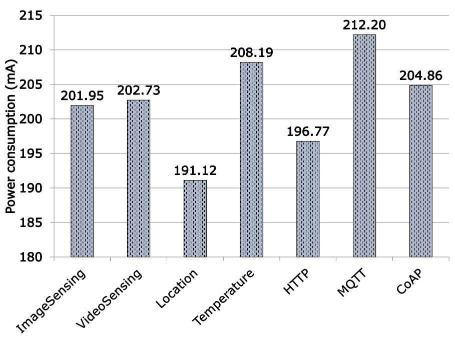
The test results indicate the need for improving the bootstrapping process in mePaaS, especially for reducing the bootstrapping time. As an alternative, the author discovered that the Node.JS based framework can reduce the burden of the bootstrapping process up to a certain level. However, most of the Android compatible modules are still in the early stage and could not just integrate into the current mePaaS framework.

Case Scenario 6.3.4. ***Performance of the process substitution***

As described in the section 4.2.4 mePaaS node can use nearby nodes for process substitution by selecting the best node for offloading the process. In this selecting process, there are two factors that would be affected by the performance of the offloading process.



(a) Time consumption for bootstrapping



(b) Average power consumption for bootstrapping

Figure 29: Service module bootstrapping cost and performance

First, the decision-making algorithm, which is based on comparing the performance and service availability of each candidate node. When the numbers of potential candidates are increasing, the algorithm should evaluate SDM from all candidates that will take the longer latency in the decision making.

Second, since the decision making requires the SDM of each candidate, the performance is also influenced by the SDM retrieval. However, this may be purely related to the fundamental wireless network protocol speed. Since the upcoming IEEE 802.11ax reaches 10 Gbps speed, this concern may be solved by the underlying hardware.

Hence, the time for the overall process can be measured based on:

$$\mathscr{T} = max\{\mathscr{T}_m^{getSDM}\} + \sum_{m \in |\mathscr{D}|} \mathscr{T}_m^{readSDM} + \mathscr{T}_m^{runAlg} \tag{6.1}$$

where:

– m denotes one candidate for offloading. The environment has a set of m, which may be denoted by M.

– $\mathscr{T}_m^{getSDM}$ is the time consumed for retrieving candidate—m's SDM asynchronously.

– $\mathscr{D}$ is a set of retrieved SDM in local memory.

– $\mathscr{T}_m^{readSDM}$ is the time consumed for reading m's SDM in local memory.

– $\mathscr{T}_m^{runAlg}$ is the time consumed for applying m in the candidate selection algorithm.

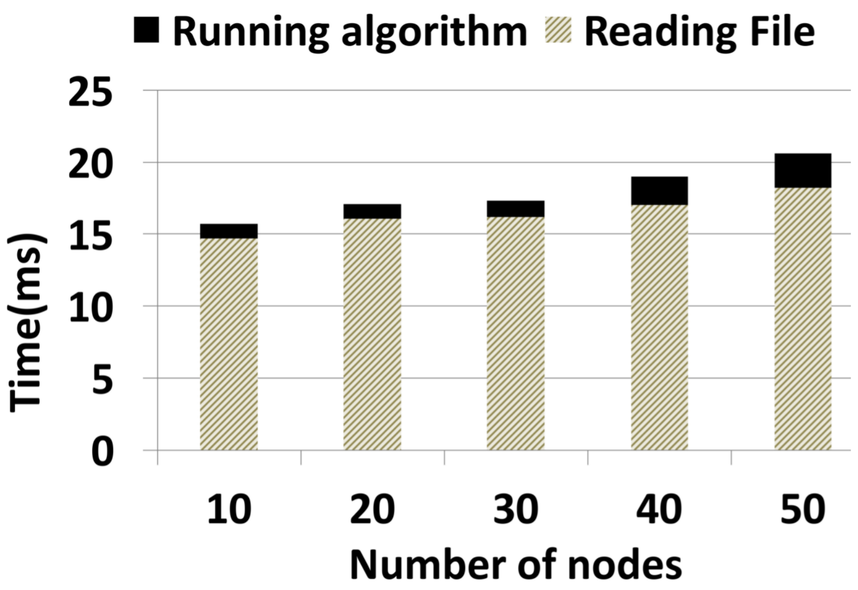


Figure 30: Average time consumption for identifying offloading node

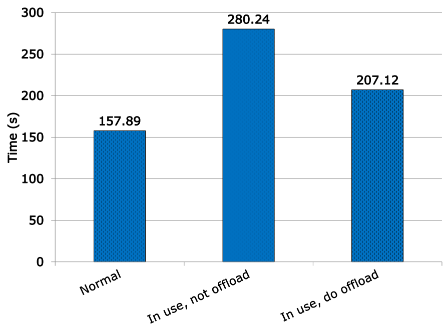
Figure 30 shows the average time consumption for identifying the best offloading node from many potential candidates, which include both reading SDM and applying the parameters from SDM to the matchmaking algorithm. It is clearly indicating that when the numbers of candidates are increasing the latency is also highly increasing due to reading all SDMs. However, the processing time does not explicitly cause much latency, which indicates that today's smartphones are quite capable of performing computational tasks.

Case Scenario 6.3.5. ***Cost and performance of the process distribution***

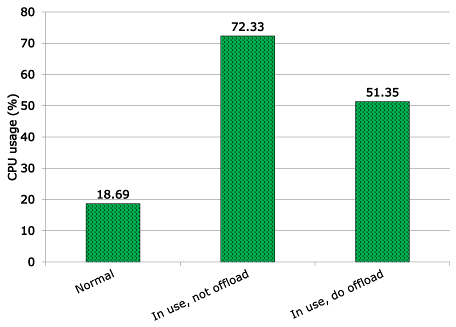
This test case evaluated the performance of the process distribution using three cases as:

- **Case 1:**The Mist node owner is not using the device while the device is performing the program from the requester that involves a video sensing task. According to the request, the mePaaS node should record a video for 30 seconds and split it into two video files and upload to a distant server. This case is mentioned as 'Normal' in Figure 31.
- **Case 2:**In this case, the Mist node owner is using the device (e.g playing a game) while the device is executing the program from the requester. Here, the author performed own CPU intensive task (average CPU usage about 41%) and at the same time do the video sensing task that was requested by the client. This case is denoted by 'In use, not offload' in Figure 31.
- **Case 3:** The Mist node owner is performing the same task and due to the lower resources (high CPU usage), the mePaaS has distributed the process

to another Mist node (use LG G4C smartphone). This case is mentioned as 'In use, do offload' in Figure 31.



(a) Average time consumption comparison



(b) Average CPU usage comparison

Figure 31: Process offloading testing results

Figure 31(a) shows the time comparison of the three cases. As the figure shows, video sensing and uploading task consumes 157.89 seconds, which includes the time for recording, splitting a video into two files, and uploading. Here we observed that still the Android system takes more time for video processing than other alternatives such as Open-CV. Moreover, still offloading consumed considerable amount of time as the Android Wi-Fi connection between the two devices was not quite fast. Figure 31(b) shows the CPU consumption comparison among the three cases. Initially, it consumes about 18% of the CPU for the given video sensing task. Next time, without offloading the CPU usage can go over 72% for the both processes. Also, we observed that sometimes the Android system kills the sensing task due to the high CPU use. However, once the mePaaS node made a decision to offload the process to the proximity mist node, the figure shows that the CPU usage has reduced to 51% that includes the video recording and offloading the task to another node.

6.3.3. Discussion

The execution of a resource intensive application on the device while the mePaaS framework operates in the background has helped evaluate the self-configured service provisioning scheme Since the resource intensive application has made the hardware resource required for certain service modules insufficient, Service Provisioning has reconfigured the Service Description Metadata automatically. Considering the process substitution performance, the result has shown that when the number of candidates increased, more SDMs need to be processed and hence the latency increases. mePass nodes are capable of distributing a process to another Mist node when it cannot perform a task by itself. As Figure 28 shows, offloading a process to another node when required enhances the performance of the service provisioning instead of performing all tasks on the device itself.For instance, offloading a task to another node reduces the CPU load from 72% to 51% and the time to complete the process also gets reduced from 280 seconds to 207 seconds.

6.4. Evaluation of Lightweight Mobile Web Service Provisioning for the Internet of Things Mediation

6.4.1. Objective

Chapter 3 presents an energy efficient, lightweight Mobile Web Service provisioning framework that utilizes technologies designed to constraint the Internet of Things environment. The framework attempts to overcome resource-intensive issues in the mobile embedded service provisioning domain; such as limited memory, processing power, a quick drain of the battery, etc. The following section provides details of the prototype implementation and testing scenarios. In addition, during the implementation and testing, RESTful architecture and lightweight protocols were used to ensure less complexity while assuring energy efficiency of the framework. The evaluation results have shown that the proposed lightweight MWS framework can provide a more cost-efficient MWS provisioning solution as against past traditional MWS frameworks.

6.4.2. Experimental Setup and Prototype Implementation

MWoT server

The MWoT server was implemented on Google/LG Nexus 5 running Android version 5.0.1. The implementation is basically adapted from the JCoAP[1] that provides a Java API for the CoAP. In addition, MWoT server has also been tested on Raspberry Pi B+ and Nexus 9 tablets.

For the external sensory data collection, the author implemented Arduino based temperature sensor module which senses the ambient temperature and sends the data to the MWoT server over the BLE connection. The Arduino setup includes the MEGA ADK board (microcontroller board based on the ATmega2560), LM35 temperature sensor and the RedBear BLE Shield (based on the Nordic nRF8001 Bluetooth Low Energy IC). The author implemented the Bluetooth communication at the Raspberry Pi with the LogiLink CSR Bluetooth v4.0 dongle, BlueZ 5.29 and Node.js.

To start an instance of the Web server, The MWoT server needs to instantiate a new CoAP local endpoint to start providing services. In the implementation, the author defined four types of resources:

- *TmpResource* - provides the current room temperature,

[1] https://github.com/dapaulid/JCoAP

- *LocationResource* - provides current location details (GPS) details of the MWoT server,
- *AltitudeResource* - provides current altitude information, and
- *LightResource* - provides the ambient light of the environment

At the present implementation, these resources are only designed to perform the GET method which is called by the clients to get services from the MWoT server.

EXI data process

The author used ExiProcessor[2] the open source Java-based library that encodes text-XML files into binary EXI and decodes EXI files back to XML. The current prototype uses pre-compressed EXI files because the current Android OS SDK does not support a number of required API libraries for ExiProcessor. For the testing, the author managed to implement ExiProcessor on the Raspberry Pi with the 3G dongle for the mobile Internet connectivity.

Local and global service discovery

Clients within the proximity can discover the MWoT server from the IP described by the BLE advertisement without establishing Bluetooth pairing connection with the MWoT server. For the global discovery, the DNS server was simulated in a regular laptop computer.

Case Scenario 6.4.1. ***Throughput comparison***

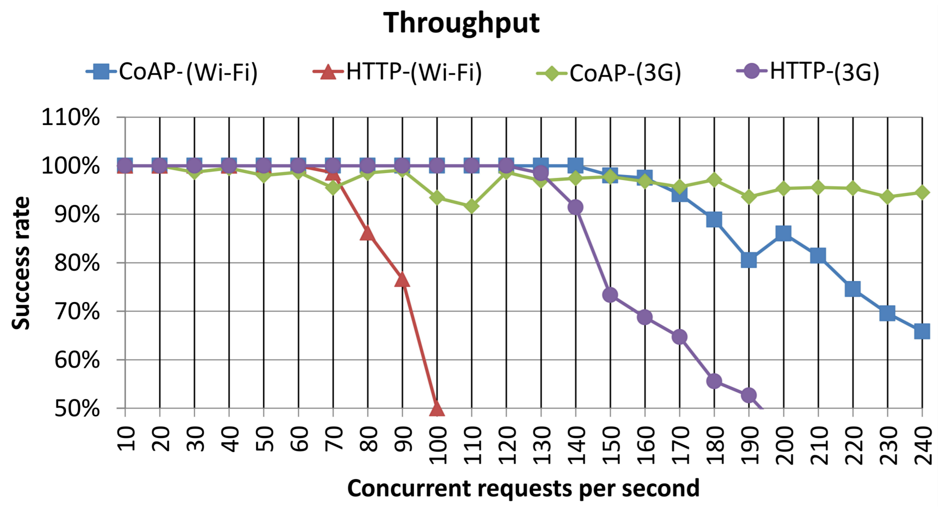


Figure 32: Throughput of MWS

Figure 32 illustrates the throughput comparison between the conventional HTTP-based MWS framework and the proposed lightweight MWS framework. The proposed framework can maintain the 100% success rate up to 140 coinciding re-

[2] http://sourceforge.net/p/exiprocessor/home/Home/

quests per second and at least maintains 95% success rate up to 160 coinciding requests per second in the Wi-Fi network. Furthermore, the author observed that improved performance in the 4G network because of the inconsistent packet delay than the Wi-Fi network. On the other hand, the conventional mobile Web server showed that it could only handle up to 70 simultaneous requests per second in the Wi-Fi network and 130 simultaneous requests per second in the 4G network with the throughput of 95%. Moreover, we can see the throughput of the conventional MWS dropped drastically after reaching its maximum capability.

Case Scenario 6.4.2. ***CPU usage comparison***

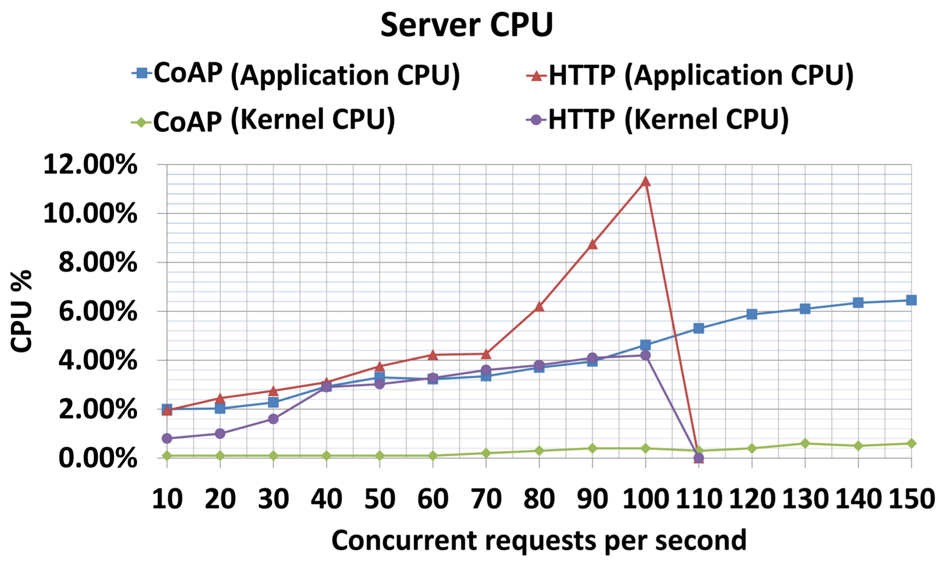
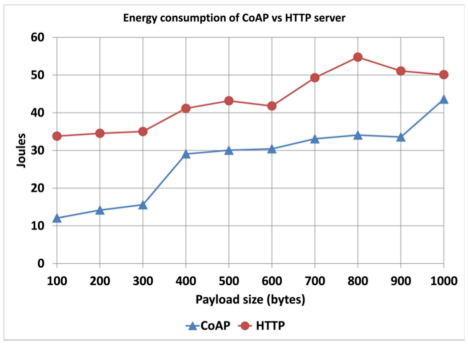


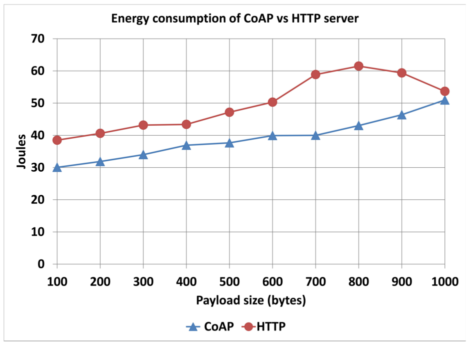
Figure 33: CPU load of MWS

In order to record the CPU usage while the MWoT server is providing services, the author utilised the Android Device Monitor that displays the CPU Load for the top applications running on the mobile device. As Figure 33 shows, the average CPU load is below 5% in the CoAP based server implementation. Conversely, in HTTP-based MWoT server implementation, it is around 11.75% for 100 simultaneous client requests per second. When the numbers of client requests are increasing (around 110/Sec), the HTTP-based MWoT server application crashed because it is CPU intensive. On the other hand, CoAP-based MWoT server can accommodate even a higher load from clients. Another interesting factor the author observed is that the kernel CPU loads of both applications. In the HTTP-based MWoT server, kernel CPU load is at a very high level while comparing with the CoAP-based MWoT server, which is as little as about 1%.

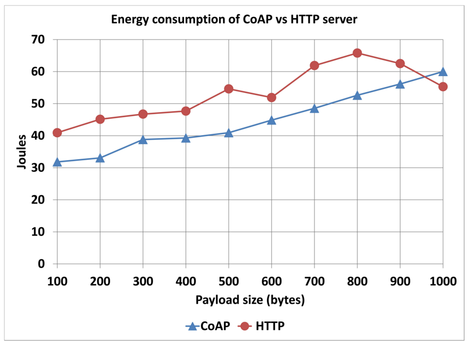
Case Scenario 6.4.3. ***Energy consumption comparison***



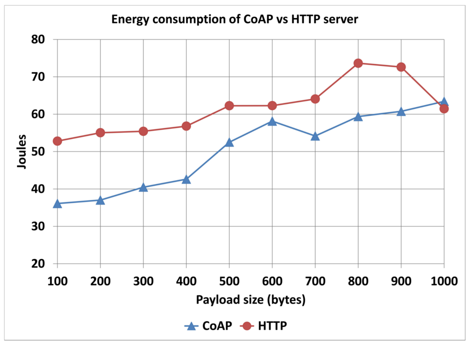
(a) 20 client requests



(b) 40 client requests



(c) 60 client requests



(d) 80 client requests

Figure 34: Comparison of energy consumption (CoAP vs HTTP)

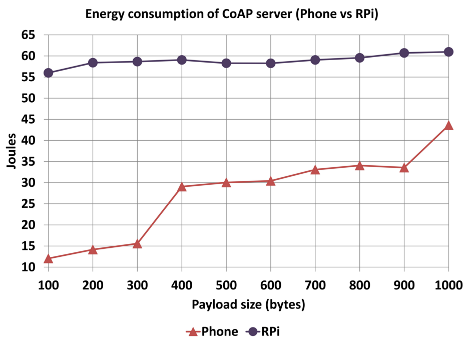
To measure the energy consumption, the author monitored total energy consumption of the device when the MWoT server is running under the different loads. During the test, the Web server serves 20, 40, 60 and 80 clients per second with the size of the payload from 100 bytes to 1000 bytes. Our test bed consists of PeakTech® Digital Multimeter which provides the visualised real-time energy consumption logging of the mobile devices. The Multimeter coupled to the battery of the phone and measures the current flow and the voltage level during the experiment. As shown in Figure 34, the CoAP server consumes less energy than the HTTP server. With 20 concurrent client requests, the average energy consumption is increased according to the size of the payload for both protocols as expected (Figure 34 Chart (a)). However, the author noticed that decrement of power consumption of the HTTP server at higher payloads (800 bytes & upwards). For instance, as shown in Figure 34. Chart (c), with 60 concurrent client requests, HTTP and CoAP consume 65.78 J and 52.61 J for 800 bytes respectively. Consequently, when the payload increases, the energy consumption increases accordingly in the CoAP server, but slightly decreases in the HTTP server. Furthermore, we can see that HTTP consumes less energy than CoAP when the payload is 1000 bytes. The reason that the author observed is many sessions have been dropped at this point due to the limitations of the HTTP server which caused the drop in energy consumption.

Case Scenario 6.4.4. ***Using a Raspberry Pi over the mobile phone***

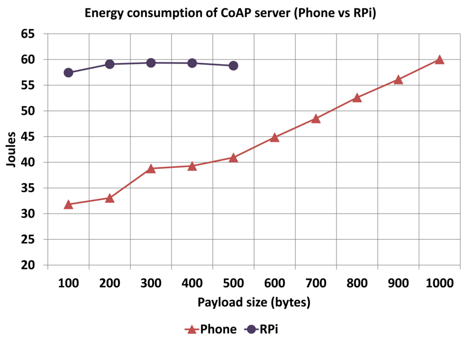
After observing the high performance of the proposed CoAP based MWoT server, the author designed a test case to investigate the performance of a mobile phone and a Raspberry Pi model B+. The author wanted to explore the feasibility of using a Raspberry Pi over the mobile phone in the MWoT service provision.

First, the author installed the CoAP Web server on the Raspberry Pi model B+ and used a 4G dongle to connect to the Internet. The author measured the energy consumption of the device in the idle state and when the server is running under a different number of client requests and payloads. The same setup is applied for the mobile phone and record the measurements as explained in the previous case study with help from the PeakTech Digital Multimeter.

According to the result (shown in Figure 35), the mobile phone consumes less energy than the Raspberry Pi (RPi). With 20 concurrent client request (Figure 35. Chart (a)), energy consumption of the phone varies in a large range according to the size of the payload (from 12 J to 45.5 J). However, RPi behaves differently because it consumes from 50 J to 60.9 J for the same amounts of payloads. There is a limitation the author observed that it could not increase the size of the payload along with the number of client requests due to the large number of packets lost at the RPi. For instance, when using the RPi server, there were a large number of packet losses on the client device when the payload beyond the 500 bytes and the energy consumption also decreased accordingly. Moreover, the author only able to measure the energy consumption at the RPi up to the payload of 500 bytes during the rest of the experiment. The overall result shows that at the RPi, the size of the payload is not that much affected to the average energy consumption because there is no big variance in consumed energy against the size of the payload (Figure 35. Chart (a) & (b)).



(a) 20 client requests



(b) 60 client requests

Figure 35: Comparison of energy consumption (Phone vs Raspberry Pi)

Case Scenario 6.4.5. ***Performance of Global Sensing Service Scheduler***

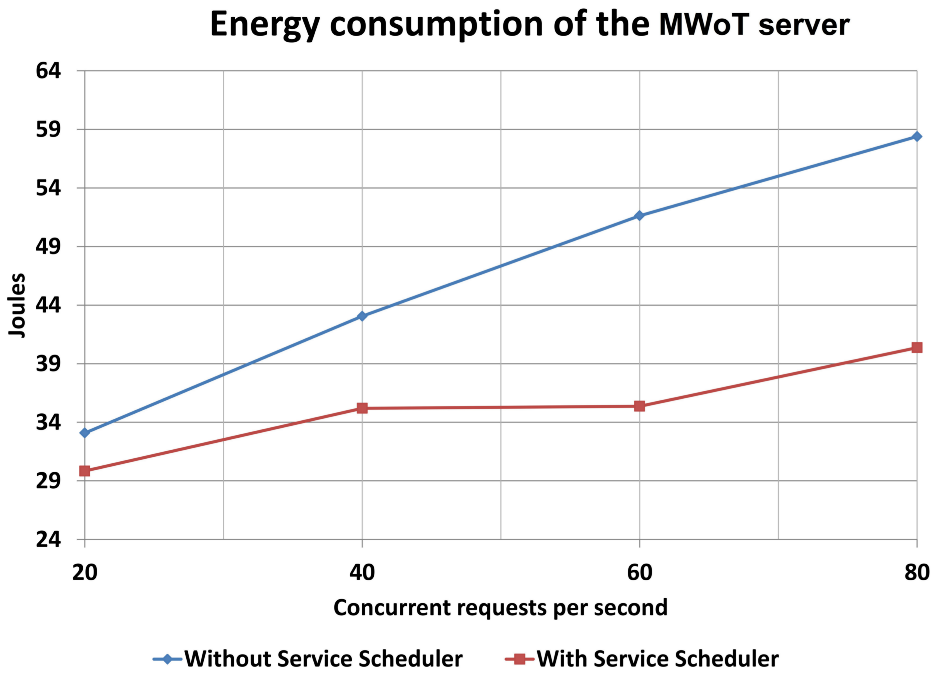


Figure 36: Comparison of energy consumption

In this experiment, the author added Global Sensing Service Scheduler into MWoT server. Then measured the energy consumption for a different number of coinciding client requests that are asking the list of temperatures in five buildings for the last six hours. The size of the payload of the reply message is around 800kb. With the Sensing Service Scheduler, MWoT server does not have to send a reply with the same data for all the clients, instead of after sending the response for the first request, it copies the data to a cloud instance and replies with the URL of the server which has the data to the subsequent requests. According to the result shown in Figure 36, the Sensing Service Scheduler keeps minimising the energy consumption by handling multiple service requests for the same resource. More details of experiments conducted with the Global Sensing Service Scheduler can be found in the work at [26].

6.4.3. Discussion

Several test cases to investigate the performance of the proposed lightweight Web service provisioning framework were designed. The first test case compares the throughput of the framework against the traditional HTTP based mobile Web server. The results shown in figure 29, reveal that the proposed framework can maintain 100% success rate up to 140 coinciding requests per second while the traditional framework can only handle about 70 concurrent requests per second. In addition, other performance evaluation results (Server CPU, Energy consumption, etc.) have shown that the proposed lightweight MWS framework can provide a more cost-efficient MWS provisioning solution as against past traditional MWS frameworks.

6.5. Summary

In this chapter, the author has presented details of the implemented prototypes and the evaluation results of the proposed frameworks.

In the first section, the author has shown that the optimised scheduling strategy can save energy for the collaborative IoT devices. The framework has been deployed on LG Sprit mobile phones running Android version 5.0.1., and the FogNets provider device has embedded with NanoHTTPd server. Then the author has designed a test case to evaluate the performance of the proposed scheduling algorithm. The results have shown that the optimised scheduling approach could increase the energy efficiency of the MWoT device. Moreover, it also has been demonstrated that that participating in the collaboration will not drain much extra energy from IoT devices that have been used as FogNets provider.

The second section presented the details of the prototype that the author has designed and implemented for the mePaaS framework on an Android OS mobile device (LG G4C). The author also described how the main components work together and how the Self-configured Service Description feature could change the SDM. After that, it has shown that an experiment to compare the communication delay between three paths as cloud, fog, and mist and confirms that the Mist nodes has the lowest communication delay. Moreover, the performance of the process distribution has shown that offloading a task to a nearby mePass node provide better performance than not offloading when processing a burdensome task.

The last section describes the implementation details of the proposed lightweight MWoT service provisioning framework on the mobile device. Then the author has presented the performance evaluation test cases that designed to compare the consumption of energy and the computational resources of the proposed framework over the traditional frameworks. The evaluation results have shown that the proposed lightweight MWoT framework can provide a more cost efficient MWoT service provisioning solution than the past conventional frameworks.

7. CONCLUSION AND FUTURE RESEARCH DIRECTION

7.1. Research Contributions

Contemporary smartphones are rapidly becoming ubiquitous because of the integrated embedded hardware and software sensor components and mobile network technologies. In addition, smartphones can perform as mobile sensing service gateways that exploit established communication techniques to interact with sensors and actuators in proximity, and provide real-time spatial information to remote servers or directly to clients via embedded mobile device-hosted web services. Given the capabilities of contemporary mobile devices, the author proposed using mobile devices as MIoT devices and, to overcome the interoperability issues across the heterogeneous platforms and applications, to use the W3C WoT standard. This standard enables seamless communication over standard web technologies and increases the sustainability of MIoT deployment. To differentiate from the general MIoT, in this thesis, the author termed the WoT-driven MIoT devices as MWoT devices.

To realise the MWoT architecture, this thesis proposed an energy-efficient, lightweight MWoT service provisioning framework in Chapter 3. The proposed framework addresses the underlying challenges of the constrained nature of available resources on MWoT devices. In this chapter, the author investigated how lightweight protocols can be integrated into MWoT devices. Using lightweight protocols, MWoT devices can provide the highest quality services, while overcoming the fundamental issues caused by constrained resources. For example, the author proposed using BLE as local service discovery, and collecting data from the sensors in close proximity. Moreover, instead of traditional HTTP, the author introduced CoAP as the application layer protocol, and reduced the message payload by using the EXI format.

However, when processing a large number of concurrent client requests, conflicts will occur between the services because the devices have limited sensing components and may be unable to operate concurrently. Moreover, some clients may seek periodical data and some may request real-time data. To overcome those limitations and provide the services in a timely manner, the author also introduced a service scheduling feature that addresses the discussed challenges to provide uninterrupted service. Moreover, the service scheduler manager provides a schedule for the MWoT devices that can provide services based on available time and resources.

Although the service scheduler manager provides a suitable approach to handle the conflict between services, it is insufficient when the MWoT device is served with a large amount of data. For instance, if one client requests a resource-extensive service task—such as a video-sensing service that consumes significant processing power and bandwidth—while another client simultaneously requests an image-processing service that is also a resource-extensive task, the whole system may become unstable and most services will be unavailable. In this situation, the system will face a lack of resources, and even the service scheduler manager cannot handle this issue.

To address this type of resource constraint issue, in Chapter 4, the author proposed a framework, called mePaaS, that can execute customised computational processes defined by their requesters, and use resources from the available nearby devices. The proposed framework uses a plugin module-based approach that optimises the usage of native computational and networking components when handling the complex type of services. The feature introduced here is defined as a self-configured service description that dynamically updates the SDM based on the availability of resources. Initially, if there are no resource-intensive services running, the mePaaS node publishes all available services with the SDM. Following this, the controller communicates with the service availability controller that is continuously communicating with the resource state monitoring component to attain the latest state of the resource usage of the service components. Thereafter, the service provisioning component publishes the latest available services in the SDM, based on the availability of resources. In addition, a mist node can form a grid-computing group centred by itself with other mist nodes because the SDM also describes the resource availability of a mist node. With this feature, when there is a situation such as a mePaaS node being unable to perform a requested task because of lack of its own resources, it can distribute the task among other mist nodes in proximity. Moreover, the author introduced a work distribution scheme that will enable one mePaaS node to offload a task to other mePaaS nodes.

In the context in which the MWoT device works as a service gateway for other IoT devices, the type of communication channels also seriously affects the energy conservation of the device. For example, it is more energy intensive to use mobile Internet-based data transmission to upload the collected data than using a Wi-Fi connection. Moreover, when the number of times in which the communication sessions are increased, the amount of the consumed energy increases accordingly. To address energy-related issues when the MWoT device is operating as a service gateway, the author introduced a proactive gateway service scheduling scheme that shares the Internet opportunistically. The proposed framework enables the MWoT device to discover the proximity using low-powered service discovery, such as BLE, while collecting data from the sensors in proximity. In the meantime, the MWoT device can discover the potential FogNets providers. After the initial setup, the MWoT device can use the low-powered communication channel to upload data via the selected FogNets. Due to various reasons, the selected FogNets providers may not be available continuously, and MWoT device may require changing the gateway accordingly. The proposed framework aims to optimise the FogNets connection schedule by using the same FogNets provider for subsequent data uploading periods that reduces the extra energy consumption derived from the switching FogNets providers. Moreover, this research indicated that a FogNets provider device does not consume much more power when uploading data within a certain limit.

7.2. Future Research Directions

- **Service provisioning/discovery**
  In Chapter 3, the MWoT system utilises standard web service publishing mechanisms such as publishing its service description metadata to a global service registry and a remote client can discover the MWoT server by requesting the service registry and, using a BLE advertisement for the local service discovery. However, the proposed protocols have some limitations such as relying on a centralised system, delays in the service discovery, lot more data in the service description, etc. One of the possible solutions will be a distributed approach to service publishing, including a mesh networking solution that can improve service discovery and visibility.
- **Server-less architecture for energy efficiency**
  Recently, the Function as a Service [55] defined a very lightweight system, combined with server-less architecture. Instead of running a complete server on a mobile device, it might be possible to merge the MWoT service provisioning framework with the server-less architecture that is mainly focused on individual functions. Future designers can use this approach to minimise the energy consumption of the MWoT service provisioning framework.
- **Quality of Experience (QoE)**
  In the proactive FogNets scheduling scheme proposed in Chapter 5, it assumes that the QoS of the available FogNets providers is the same. However, in reality, the QoE attributes of those FogNets providers maybe not the same. For example, when considering a FogNets provider with more connected client nodes with limited bandwidth may be not performing well compared to another FogNets provider that has a high bandwidth and less number of connected client nodes. Consequently, another potential future work is to apply the QoE attributes such as the bandwidth, the number of connected nodes, queue length, etc. to the process of the scheduling algorithm to enhance the selection procedure.
- **Security and Privacy**
  Currently, we did not consider security and privacy issues in mobile Web service provisioning. However, it will be of great interest to add confidentiality and security levels to the current framework. In particular, the privacy of data, such as healthcare data, the ability to detect malicious packets or activities and ignore them, can be considered to enhance the security of service provisioning.

- **Formal methods/mathematical proofs for the framework**
  In this thesis, only the experimental proofs have been applied to evaluate the proposed frameworks. However, for future developers, it will be interesting to apply a mathematical model to confirm the energy efficiency of the MWoT service provisioning framework. There are numerous approaches such as eDiscovery [64], an energy efficient adaptive device discovery protocol, deep Q-learning model [4], which enhance the efficiency in computation offloading, a deep-learning-based response-time-prediction framework [5] etc. These works can be a starting point for formal proofs that can be applied for the future design of MWoT framework.

BIBLIOGRAPHY

[1] Mohammad Aazam and Eui-Nam Huh. Fog computing and smart gateway based communication for cloud of things. In *Future Internet of Things and Cloud (FiCloud), 2014 International Conference on*, pages 464–470. IEEE, 2014.

[2] Bilal Afzal, Sheeraz A Alvi, Ghalib A Shah, and Waqar Mahmood. Energy efficient context aware traffic scheduling for iot applications. *Ad Hoc Networks*, 62:101–115, 2017.

[3] Ala Al-Fuqaha, Mohsen Guizani, Mehdi Mohammadi, Mohammed Aledhari, and Moussa Ayyash. Internet of things: A survey on enabling technologies, protocols, and applications. *Communications Surveys & Tutorials, IEEE*, 17(4):2347–2376, 2015.

[4] Md Golam Rabiul Alam, Mohammad Mehedi Hassan, Md ZIa Uddin, Ahmad Almogren, and Giancarlo Fortino. Autonomic computation offloading in mobile edge for iot applications. *Future Generation Computer Systems*, 90:149–157, 2019.

[5] Abdulhameed Alelaiwi. An efficient method of computation offloading in an edge cloud platform. *Journal of Parallel and Distributed Computing*, 2019.

[6] Mushtaq Ali, Mohamad Fadli Zolkipli, Jasni Mohamad Zain, and Shahid Anwar. Mobile cloud computing with soap and rest web services. In *Journal of Physics: Conference Series*, volume 1018, page 012005. IOP Publishing, 2018.

[7] Feda AlShahwan and Maha Faisal. Mobile cloud computing for providing complex mobile web services. In *Mobile Cloud Computing, Services, and Engineering (MobileCloud), 2014 2nd IEEE International Conference on*, pages 77–84. IEEE, 2014.

[8] Feda AlShahwan and Klaus Moessner. Providing soap web services and restful web services from mobile hosts. In *Internet and Web Applications and Services (ICIW), 2010 Fifth International Conference on*, pages 174–179. IEEE, 2010.

[9] Giuseppe Anastasi, Marco Conti, Mario Di Francesco, and Andrea Passarella. Energy conservation in wireless sensor networks: A survey. *Ad Hoc Networks*, 7(3):537–568, 2009.

[10] Soumya Kanti Datta Arne Broring. Web of things - technology landscape. `https://w3c.github.io/wot/landscape.html`, 03 2018. (Accessed on 05/25/2018).

[11] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer networks*, 54(15):2787–2805, 2010.

[12] Niranjan Balasubramanian, Aruna Balasubramanian, and Arun Venkataramani. Energy Consumption in Mobile Phones: A Measurement Study and Implications for Network Applications. In *the 9th ACM SIGCOMM Conference on Internet Measurement Conference*, pages 280–293. ACM, 2009.

[13] Andrew Banks and Rahul Gupta. Mqtt version 3.1.1. `http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html`, 2014. (Accessed on 06/28/2018).

[14] Akram Bayat, Marc Pomplun, and Duc A Tran. A study on human activity recognition using accelerometer data from smartphones. *Procedia Computer Science*, 34:450–457, 2014.

[15] Stefan Berger, Scott McFaddin, Chandrasekhar Narayanaswami, and Mandayam Raghunath. Web services on mobile devices-implementation and experience. In *Mobile Computing Systems and Applications, 2003. Proceedings. Fifth IEEE Workshop on*, pages 100–109. IEEE, 2003.

[16] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific american*, 284(5):34–43, 2001.

[17] Igor Bisio, Fabio Lavagetto, Mario Marchese, and Andrea Sciarrone. Smartphone-centric ambient assisted living platform for patients suffering from co-morbidities monitoring. *IEEE Communications Magazine*, 53(1):34–41, 2015.

[18] Michael Blackstock and Rodger Lea. Toward interoperability in a web of things. In *Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication*, pages 1565–1574. ACM, 2013.

[19] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16. ACM, 2012.

[20] Joseph Bradley, Joel Barbier, and Doug Handler. Embracing the Internet of everything to capture your share of $14.4 trillion. *White Paper, Cisco*, 2013.

[21] Niels Brouwers and Koen Langendoen. Pogo, a middleware for mobile phone sensing. In *Proceedings of the 13th International Middleware Conference*, pages 21–40. Springer-Verlag New York, Inc., 2012.

[22] Angelo P Castellani, Mattia Gheda, Nicola Bui, Michele Rossi, and Michele Zorzi. Web Services for the Internet of Things through CoAP and EXI. In *Communications Workshops (ICC), 2011 IEEE International Conference on*, pages 1–6. IEEE, 2011.

[23] Chii Chang, Sea Ling, and Shonali Krishnaswamy. Promws: Proactive mobile web service provision using context-awareness. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2011 IEEE International Conference on*, pages 69–74. IEEE, 2011.

[24] Chii Chang, Sea Ling, and Satish Srirama. Trustworthy service discovery for mobile social network in proximity. In *PerCom '14 Workshops*, pages 478–483. IEEE, 2014.

[25] Chii Chang, S.W. Loke, Hai Dong, F. Salim, S.N. Srirama, M. Liyanage, and Sea Ling. An energy-efficient inter-organizational wireless sensor data collection framework. In *Proceedings of the 2015 IEEE International Conference on Web Services (ICWS)*, pages 639–646, June 2015.

[26] Chii Chang, S. N. Srirama, and Mohan Liyanage. A Service-Oriented Mobile Cloud Middleware Framework for Provisioning Mobile Sensing as a Service. In *Parallel and Distributed Systems (ICPADS 2015) , 21st IEEE International Conference on*. IEEE, 2015.

[27] Chii Chang, S N Srirama, and J Mass. A Middleware for Discovering Proximity-Based Service-Oriented Industrial Internet of Things. In *Proceedings of the 2015 IEEE International Conference on Services Computing (SCC)*, pages 130–137, jun 2015.

[28] Chii Chang, Satish Narayana Srirama, and Rajkumar Buyya. Indie fog: An efficient fog-computing infrastructure for the internet of things. *Computer*, 50(9):92–98, 2017.

[29] Chii Chang, Satish Narayana Srirama, and Rajkumar Buyya. Mobile cloud business process management system for the internet of things: a survey. *ACM Computing Surveys (CSUR)*, 49(4):70, 2017.

[30] Chii Chang, Satish Narayana Srirama, and Sea Ling. An adaptive mediation framework for mobile p2p social content sharing. In *International Conference on Service-Oriented Computing*, pages 374–388. Springer, 2012.

[31] Chii Chang, Satish Narayana Srirama, and Sea Ling. SPiCa: a social private cloud computing application framework. In *Proceedings of the 13th International Conference on Mobile and Ubiquitous Multimedia*, pages 30–39. ACM, 2014.

[32] Chii Chang, Satish Narayana Srirama, and Sea Ling. Mobile social network in proximity: taxonomy, approaches and open challenges. *International Journal of Pervasive Computing and Communications*, 11(1):77–101, 2015.

[33] Hongju Cheng, Ronglie Guo, Zhihuang Su, Naixue Xiong, and Wenzhong Guo. Service-oriented node scheduling schemes with energy efficiency in wireless sensor networks. *International Journal of Distributed Sensor Networks*, 2014.

[34] Hongju Cheng, Zhihuang Su, Naixue Xiong, and Yang Xiao. Energy-efficient node scheduling algorithms for wireless sensor networks using markov random field model. *Information Sciences*, 329:461–477, 2016.

[35] Mung Chiang. Fog networking: An overview on research opportunities. *arXiv preprint arXiv:1601.00835*, 2016.

[36] John Chon and Hojung Cha. Lifemap: A smartphone-based context provider for location-based services. *IEEE Pervasive Computing*, 10(2):58–67, 2011.

[37] Carmela Comito, Deborah Falcone, Domenico Talia, and Paolo Trunfio. Energy-aware task allocation for small devices in wireless networks. *Concurrency and Computation: Practice and Experience*, 29(1), 2017.

[38] World Wide Web Consortium et al. Json-ld 1.0: a json-based serialization for linked data. *W3C Recommendation*, 2014.

[39] Marco Conti, Chiara Boldrini, Salil S. Kanhere, Enzo Mingozzi, Elena Pagani, Pedro M. Ruiz, and Mohamed Younis. From MANET to people-centric networking: Milestones and open research challenges. *Computer Communications*, 71:1–21, 2015.

[40] Marco Conti, Sajal K Das, Chatschik Bisdikian, Mohan Kumar, Lionel M Ni, Andrea Passarella, George Roussos, Gerhard Tröster, Gene Tsudik, and Franco Zambonelli. Looking ahead in pervasive computing: Challenges and opportunities in the era of cyber–physical convergence. *Pervasive and Mobile Computing*, 8(1):2–21, 2012.

[41] Luis Corral, Anton B Georgiev, Alberto Sillitti, and Giancarlo Succi. A method for characterizing energy consumption in android smartphones. In *Proceedings of the 2nd International Workshop on Green and Sustainable Software*, pages 38–45. IEEE Press, 2013.

[42] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. Maui: making smartphones last longer with code offload. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 49–62. ACM, 2010.

[43] Tej Tharang Dandala, Vallidevi Krishnamurthy, and Rajan Alwan. Internet of vehicles (iov) for traffic management. In *2017 International Conference on Computer, Communication and Signal Processing (ICCCSP)*, pages 1–4. IEEE, 2017.

[44] Kashif Dar, Amir Taherkordi, Harun Baraki, Frank Eliassen, and Kurt Geihs. A resource oriented integration architecture for the Internet of Things: A business process perspective. *Pervasive and Mobile Computing*, 20:145–159, 2015.

[45] Joseph DeCuir. Introducing bluetooth smart: Part 1: A look at both classic and new technologies. *IEEE Consumer Electronics Magazine*, 3(1):12–18, 2014.

[46] Pratikkumar Desai, Amit Sheth, and Pramod Anantharam. Semantic gateway as a service architecture for iot interoperability. In *Mobile Services*

(MS), 2015 IEEE International Conference on, pages 313–319. IEEE, IEEE Press, 2015.

[47] Charalampos Doukas, Luca Capra, Fabio Antonelli, Erinda Jaupaj, Andrei Tamilin, and Iacopo Carreras. Providing generic support for iot and m2m for mobile devices. In *Computing & Communication Technologies-Research, Innovation, and Vision for the Future (RIVF), 2015 IEEE RIVF International Conference on*, pages 192–197. IEEE, 2015.

[48] Adam Dunkels. Full tcp/ip for 8-bit architectures. In *Proceedings of the 1st international conference on Mobile systems, applications and services*, pages 85–98. ACM, 2003.

[49] Adam Dunkels, Bjorn Gronvall, and Thiemo Voigt. Contiki-a lightweight and flexible operating system for tiny networked sensors. In *Local Computer Networks, 2004. 29th Annual IEEE International Conference on*, pages 455–462. IEEE, 2004.

[50] Robert Elfwing, Ulf Paulsson, and Lars Lundberg. Performance of SOAP in Web Service environment compared to CORBA. In *Software Engineering Conference, 2002. Ninth Asia-Pacific*, pages 84–93. IEEE, 2002.

[51] Sinem Coleri Ergen and Pravin Varaiya. Tdma scheduling algorithms for wireless sensor networks. *Wireless Networks*, 16(4):985–997, 2010.

[52] Dave Evans. The internet of things: How the next evolution of the internet is changing everything. *CISCO white paper*, 1:14, 2011.

[53] Niroshinie Fernando, Seng W Loke, and Wenny Rahayu. Honeybee: A programming framework for mobile crowd computing. In *International Conference on Mobile and Ubiquitous Systems: Computing, Networking, and Services*, pages 224–236. Springer, 2012.

[54] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, 2000.

[55] Geoffrey C Fox, Vatche Ishakian, Vinod Muthusamy, and Aleksander Slominski. Status of serverless computing and function-as-a-service (faas) in industry and research. *arXiv preprint arXiv:1708.08028*, 2017.

[56] Guido Gehlen and Linh Pham. Mobile web services for peer-to-peer applications. In *Consumer Communications and Networking Conference, 2005. CCNC. 2005 Second IEEE*, pages 427–433. IEEE, 2005.

[57] Tuan Nguyen Gia, Imed Ben Dhaou, Mai Ali, Amir M Rahmani, Tomi Westerlund, Pasi Liljeberg, and Hannu Tenhunen. Energy efficient fog-assisted iot system for monitoring diabetic patients with cardiovascular disease. *Future Generation Computer Systems*, 93:198–211, 2019.

[58] Tuan Nguyen Gia, Mingzhe Jiang, Amir-Mohammad Rahmani, Tomi Westerlund, Pasi Liljeberg, and Hannu Tenhunen. Fog computing in healthcare internet of things: A case study on ecg feature extraction. In *Computer and Information Technology; Ubiquitous Computing and Communi-*

cations; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing (CIT/IUCC/DASC/PICOM), 2015 IEEE International Conference on, pages 356–363. IEEE, 2015.

[59] Carles Gomez, Joaquim Oller, and Josep Paradells. Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology. *Sensors*, 12(9):11734–11753, 2012.

[60] Dominique Guinard. *A web of things application architecture: Integrating the real-world into the web*. PhD thesis, ETH Zurich, 2011.

[61] Dominique Guinard, Vlad Trifa, Stamatis Karnouskos, Patrik Spiess, and Domnic Savio. Interacting with the soa-based internet of things: Discovery, query, selection, and on-demand provisioning of web services. *IEEE transactions on Services Computing*, 3:223–235, 2010.

[62] Dominique Guinard, Vlad Trifa, Friedemann Mattern, and Erik Wilde. From the internet of things to the web of things: Resource-oriented architecture and best practices. In *Architecting the Internet of things*, pages 97–129. Springer, 2011.

[63] Dominique Guinard, Vlad Trifa, and Erik Wilde. A resource oriented architecture for the web of things. In *Internet of Things (IOT), 2010*, pages 1–8. IEEE, 2010.

[64] Bo Han, Jian Li, and Aravind Srinivasan. On the energy efficiency of device discovery in mobile opportunistic networks: A systematic approach. *IEEE Transactions on Mobile Computing*, 14(4):786–799, 2015.

[65] Dae-Man Han and Jae-Hyun Lim. Design and implementation of smart home energy management systems based on zigbee. *IEEE Transactions on Consumer Electronics*, 56(3), 2010.

[66] W R Heinzelman, A Chandrakasan, and H Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *System Sciences, 2000. Proceedings of the 33rd Annual Hawaii International Conference on*, pages 10 pp. vol.2–. IEEE Press, 2000.

[67] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System architecture directions for networked sensors. *ACM SIGOPS operating systems review*, 34(5):93–104, 2000.

[68] Jonathan W Hui and David E Culler. Extending ip to low-power, wireless personal area networks. *IEEE Internet Computing*, 12(4), 2008.

[69] IEEE. Ieee sa - fog - fog computing and networking architecture framework. `https://standards.ieee.org/develop/wg/FOG.html`, 2008. (Accessed on 05/25/2018).

[70] ITU-T. Y.2069. terms and definitions for the internet of things. `http://www.itu.int/rec/T-REC-Y.2069-201207-I/en`, July 2012. (Accessed on 05/23/2018).

[71] Marc Jansen, O Hordt, and Jelena Milatovic. About the development of scenarios for mobile Web Service provisioning. In *Computer and Information Technology (WCCIT), 2013 World Congress on*, pages 1–6. IEEE, 2013.

[72] Yaser Jararweh, Ahmad Doulat, Omar AlQudah, Ejaz Ahmed, Mahmoud Al-Ayyoub, and Elhadj Benkhelifa. The future of mobile cloud computing: integrating cloudlets and mobile edge computing. In *Telecommunications (ICT), 2016 23rd International Conference on*, pages 1–5. IEEE, 2016.

[73] Schneider John, Kamiya Takuki, Peintner Daniel, and Kyusakov Rumen. Efficient xml interchange (exi) format 1.0 (second edition). `https://www.w3.org/TR/2014/REC-exi-20140211/`, February 2014. (Accessed on 07/04/2018).

[74] Antero Juntunen, Sakari Luukkainen, and Virpi Kristiina Tuunainen. Deploying nfc technology for mobile ticketing services–identification of critical business model issues. In *Mobile Business and 2010 Ninth Global Mobility Roundtable (ICMB-GMR), 2010 Ninth International Conference on*, pages 82–90. IEEE, 2010.

[75] Omprakash Kaiwartya, Abdul Hanan Abdullah, Yue Cao, Ayman Altameem, Mukesh Prasad, Chin-Teng Lin, and Xiulei Liu. Internet of vehicles: Motivation, layered architecture, network model, challenges, and future aspects. *IEEE Access*, 4:5356–5373, 2016.

[76] Stephan Karpischek, Florian Michahelles, Florian Resatsch, and Elgar Fleisch. Mobile sales assistant-an nfc-based product information system for retailers. In *Near Field Communication, 2009. NFC'09. First International Workshop on*, pages 20–23. IEEE, 2009.

[77] Johannes Hund Kazuo Kajimoto, Ryuichi Matsukura. Web of things (wot) architecture. `https://w3c.github.io/wotwg/architecture/wot-architecture.html`, February 2018. (Accessed on 05/25/2018).

[78] Il Kon Kim and Joon Hyun Song. Mobile Health reference architectures. In *Information Society (i-Society), 2013 International Conference on*, pages 158–164. IEEE, 2013.

[79] Jong-Myoung Kim, Seon-Ho Park, Young-Ju Han, and Tai-Myoung Chung. CHEF: cluster head election mechanism using fuzzy logic in wireless sensor networks. In *Advanced communication technology, 2008. ICACT 2008. 10th international conference on*, volume 1, pages 654–659. IEEE, IEEE Press, 2008.

[80] Yeon-Seok Kim and Kyong-Ho Lee. A lightweight framework for mobile web services. *Computer Science-Research and Development*, 24(4):199–209, 2009.

[81] Derrick Kondo, Bahman Javadi, Paul Malecot, Franck Cappello, and David P Anderson. Cost-benefit analysis of cloud computing versus desk-

top grids. In *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pages 1–12. IEEE, 2009.

[82] Gerd Kortuem, Fahim Kawsar, Vasughi Sundramoorthy, and Daniel Fitton. Smart objects as building blocks for the internet of things. *IEEE Internet Computing*, 14(1):44–51, 2010.

[83] Nandakishore Kushalnagar, Gabriel Montenegro, and Christian Schumacher. Ipv6 over low-power wireless personal area networks (6lowpans): overview, assumptions, problem statement, and goals. Technical report, RFC 4919, 2007.

[84] Yiping Li, Xiaotong Zhang, Jun Zeng, Yadong Wan, and Fuqiang Ma. A distributed tdma scheduling algorithm based on energy-topology factor in internet of things. *IEEE Access*, 5:10757–10768, 2017.

[85] Xingqin Lin, Jeffrey Andrews, Amitabha Ghosh, and Rapeepat Ratasuk. An overview of 3gpp device-to-device proximity services. *Communications Magazine, IEEE*, 52(4):40–48, 2014.

[86] Zhao Liqiang, Yin Shouyi, Liu Leibo, Zhang Zhen, and Wei Shaojun. A crop monitoring system based on wireless sensor network. *Procedia Environmental Sciences*, 11:558–565, 2011.

[87] Jen-Hao Liu, Yu-Fan Chen, Tzu-Shiang Lin, Da-Wei Lai, Tzai-Hung Wen, Chih-Hong Sun, Jehn-Yih Juang, and Joe-Air Jiang. Developed urban air quality monitoring system based on wireless sensor networks. In *Sensing technology (icst), 2011 fifth international conference on*, pages 549–554. IEEE, 2011.

[88] M Liyanage, Chii Chang, and S N Srirama. Lightweight Mobile Web Service Provisioning for Sensor Mediation. In *Mobile Services (MS), 2015 IEEE International Conference on*, pages 57–64, jun 2015.

[89] Mohan Liyanage, Chii Chang, and Satish Narayana Srirama. Energy-efficient mobile data acquisition using opportunistic internet of things gateway services. In *Internet of Things (iThings) , 2016 IEEE International Conference on*, pages 217–222. IEEE, 2016.

[90] Mohan Liyanage, Chii Chang, and Satish Narayana Srirama. mepaas: Mobile-embedded platform as a service for distributing fog computing to edge nodes. In *Parallel and Distributed Computing, Applications and Technologies (PDCAT), 2016 17th International Conference on*, pages 73–80. IEEE, 2016.

[91] Mohan Liyanage, Chii Chang, and Satish Narayana Srirama. Adaptive mobile web server framework for mist computing in the internet of things. *International Journal of Pervasive Computing and Communications. ISSN: 1742-7371,*, 14(3/4):247–267, 2018.

[92] Alexander G Logan, Warren J McIsaac, Andras Tisler, M Jane Irvine, Allison Saunders, Andrea Dunai, Carlos A Rizo, Denice S Feig, Melinda

Hamill, Mathieu Trudel, et al. Mobile phone–based remote patient monitoring system for management of hypertension in diabetic patients. *American journal of hypertension*, 20(9):942–948, 2007.

[93] Seng W Loke, Keegan Napier, Abdulaziz Alali, Niroshinie Fernando, and Wenny Rahayu. Mobile Computations with Surrounding Devices: Proximity Sensing and MultiLayered Work Stealing. *ACM Trans. Embed. Comput. Syst.*, 14(2):22:1—-22:25, feb 2015.

[94] Richard K Lomotey, Yiding Chai, Kazi A Ahmed, and Ralph Deters. Web Services Mobile Application for Geographically Dispersed Crop Farmers. In *Computational Science and Engineering (CSE), 2013 IEEE 16th International Conference on*, pages 151–158. IEEE, 2013.

[95] Richard K Lomotey and Ralph Deters. Sensor data propagation in mobile hosting networks. In *2015 IEEE Symposium on Service-Oriented System Engineering*, pages 98–106. IEEE, 2015.

[96] Richard K Lomotey, Shomoyita Jamal, and Ralph Deters. SOPHRA: a mobile web services hosting infrastructure in mHealth. In *Mobile Services (MS), 2012 IEEE First International Conference on*, pages 88–95. IEEE, 2012.

[97] Sujith Samuel Mathew, Yacine Atif, Quan Z Sheng, and Zakaria Maamar. Web of things: Description, discovery and integration. In *2011 IEEE International Conferences on Internet of Things, and Cyber, Physical and Social Computing*, pages 9–15. IEEE, 2011.

[98] Mukhtiar Memon, Stefan Rahr Wagner, Christian Fischer Pedersen, Femina Hassan Aysha Beevi, and Finn Overgaard Hansen. Ambient assisted living healthcare frameworks, platforms, standards, and quality attributes. *Sensors*, 14(3):4312–4341, 2014.

[99] KamalEldin Mohamed and Duminda Wijesekera. A lightweight framework for web services implementations on mobile devices. In *Mobile Services (MS), 2012 IEEE First International Conference on*, pages 64–71. IEEE, 2012.

[100] Prashanth Mohan, Venkata N Padmanabhan, and Ramachandran Ramjee. Nericell: rich monitoring of road and traffic conditions using mobile smartphones. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 323–336. ACM, 2008.

[101] R. Morabito, R. Petrolo, V. Loscri, and N. Mitton. Enabling a lightweight edge gateway-as-a-service for the internet of things. In *2016 7th International Conference on the Network of the Future, NOF 2016*, pages 1–5. IEEE Press, 2017.

[102] Guido Moritz, Frank Golatowski, Christian Lerche, and Dirk Timmermann. Beyond 6LoWPAN: Web services in wireless sensor networks. *Industrial Informatics, IEEE Transactions on*, 9(4):1795–1805, 2013.

[103] Suman Nath and Phillip B Gibbons. Communicating via fireflies: geographic routing on duty-cycled sensors. In *Information Processing in Sensor Networks, 2007. IPSN 2007. 6th International Symposium on*, pages 440–449. IEEE, 2007.

[104] Joseph J Oresko, Zhanpeng Jin, Jun Cheng, Shimeng Huang, Yuwen Sun, Heather Duschl, and Allen C Cheng. A wearable smartphone-based platform for real-time cardiovascular disease detection via electrocardiogram processing. *IEEE Transactions on Information Technology in Biomedicine*, 14(3):734–740, 2010.

[105] Gabriel Orsini, Dirk Bade, and Winfried Lamersdorf. Computing at the mobile edge: Designing elastic android applications for computation offloading. In *IFIP Wireless and Mobile Networking Conference (WMNC), 2015 8th*, pages 112–119. IEEE, 2015.

[106] Zhonghong Ou, Meina Song, Hui Chen, and Junde Song. Layered peer-to-peer architecture for mobile web services via converged cellular and ad hoc networks. In *Grid and Pervasive Computing Workshops, 2008. GPC Workshops' 08. The 3rd International Conference on*, pages 195–200. IEEE, 2008.

[107] Carlos Paniagua. *Discovery and push notification mechanisms for mobile cloud services*. PhD thesis, Master's thesis, University of Tartu, 2012.

[108] Kwanghyo Park, Hyojeong Shin, and Hojung Cha. Smartphone-based pedestrian tracking in indoor corridor environments. *Personal and Ubiquitous Computing*, 17(2):359–370, 2013.

[109] Milan Patel, B Naughton, C Chan, N Sprecher, S Abeta, A Neal, et al. Mobile-edge computing introductory technical white paper. *White Paper, Mobile-edge Computing (MEC) industry initiative*, 2014.

[110] Charith Perera, Prem Prakash Jayaraman, Arkady Zaslavsky, Dimitrios Georgakopoulos, and Peter Christen. Mosden: An internet of things middleware for resource constrained mobile devices. In *System Sciences (HICSS), 2014 47th Hawaii International Conference on*, pages 1053–1062. IEEE, 2014.

[111] Charith Perera, Arkady Zaslavsky, Peter Christen, and Dimitrios Georgakopoulos. Context aware computing for the internet of things: A survey. *Communications Surveys & Tutorials, IEEE*, 16(1):414–454, 2014.

[112] Jürgo S Preden, Kalle Tammemäe, Axel Jantsch, Mairo Leier, Andri Riid, and Emine Calis. The benefits of self-awareness and attention in fog and mist computing. *Computer*, 48(7):37–45, 2015.

[113] Petri Pulli, Olli Martikainen, Ye Zhang, Valeriy Naumov, Zeeshan Asghar, and Antti Pitkänen. Augmented processes: A case study in healthcare. In *Proceedings of the 4th International Symposium on Applied Sciences in Biomedical and Communication Technologies*, page 137. ACM, 2011.

[114] Xiao Qin, Hong Jiang, Yifeng Zhu, and David R Swanson. Dynamic load balancing for i/o-intensive tasks on heterogeneous clusters. In *High Performance Computing-HiPC 2003*, pages 300–309. Springer, 2003.

[115] Brian Raymor, Bill Silverajan, Carsten Bormann, Klaus Hartke, Hannes Tschofenig, and Simon Lemay. Rfc 8323 - coap (constrained application protocol) over tcp, tls, and websockets. `https://datatracker.ietf.org/doc/rfc8323/`, 2018. (Accessed on 06/28/2018).

[116] R. Robinson. Understand enterprise service bus scenarios and solutions in service-oriented architecture, part 1: The role of the enterprise service bus. `http://www.ibm.com/developerworks/webservices/library/ws-esbscen/`, 2004.

[117] Peter Saint-Andre. Streaming xml with jabber/xmpp. *IEEE internet computing*, 9(5):82–89, 2005.

[118] João Santos, Joel J P C Rodrigues, Bruno M C Silva, João Casal, Kashif Saleem, and Victor Denisov. An IoT-based mobile gateway for intelligent personal assistants on mobile health environments. *Journal of Network and Computer Applications*, 71:194–204, 2016.

[119] Peramanathan Sathyamoorthy, Edith C-H Ngai, Xiping Hu, and Victor Leung. Profiling energy efficiency and data communications for mobile internet of things. *Wireless Communications and Mobile Computing*, 2017, 2017.

[120] Mahadev Satyanarayanan, Victor Bahl, Ramón Caceres, and Nigel Davies. The case for vm-based cloudlets in mobile computing. *IEEE pervasive Computing*, 2009.

[121] Jochen Schiller, Achim Liers, and Hartmut Ritter. Scatterweb: A wireless sensornet platform for research and teaching. *Computer Communications*, 28(13):1545–1551, 2005.

[122] Takuki Kamiya Sebastian Kaebisch. Web of things (wot) thing description. `https://www.w3.org/TR/wot-thing-description/`, 4 2018. (Accessed on 06/07/2018).

[123] Seyyit Alper Sert, Hakan Bagci, and Adnan Yazici. Mofca: Multi-objective fuzzy clustering algorithm for wireless sensor networks. *Applied Soft Computing*, 30:151–165, 2015.

[124] Eyal M Sharon, Jed Stremel, Olumakinde A Adeagbo, Wayne Chang, Joseph Hewitt, and Matthew Cahill. Sharing of location-based content item in social networking service, August 25 2015. US Patent 9,119,027.

[125] Zach Shelby. Embedded web services. *IEEE Wireless Communications*, 17(6):52, 2010.

[126] Zach Shelby. Rfc 6690 - constrained restful environments (core) link format. `https://tools.ietf.org/html/rfc6690.html`, 2012. (Accessed on 06/28/2018).

[127] Zach Shelby, Carsten Bormann, and Srdjan Krco. Core resource directory. *IETF*, 2013. (Accessed on 07/02/2018).

[128] Zach Shelby, Brian Frank, and Don Sturek. Constrained Application Protocol (CoAP) draft-shelby-core-coap-00. *Online at http://tools. ietf. org/html/draft-shelby-core-coap-01*, 2010.

[129] Zach Shelby, Michael Koster, Carsten Bormann, Peter Van der Stok, and Christian Amsuess. CoRE Resource Directory. Internet-Draft draft-ietf-core-resource-directory-15, Internet Engineering Task Force, oct 2018. (Accessed on 10/18/2018).

[130] Bluetooth SIG. Bluetooth specification version 4.0. *Bluetooth SIG*, 2010.

[131] Cristiano M Silva, Fabricio A Silva, João FM Sarubbi, Thiago R Oliveira, Wagner Meira Jr, and Jose Marcos S Nogueira. Designing mobile content delivery networks for the internet of vehicles. *Vehicular communications*, 8:45–55, 2017.

[132] Olena Skarlat, Stefan Schulte, Michael Borkowski, and Philipp Leitner. Resource provisioning for iot services in the fog. In *2016 IEEE 9th Conference on Service-Oriented Computing and Applications (SOCA)*, pages 32–39. IEEE, 2016.

[133] Arun A Somasundara, Aditya Ramamoorthy, and Mani B Srivastava. Mobile element scheduling for efficient data collection in wireless sensor networks with dynamic deadlines. In *Real-Time Systems Symposium, 2004. Proceedings. 25th IEEE International*, pages 296–305. IEEE, 2004.

[134] Satish Srirama and Wolfgang Prinz. Concept, implementation and performance testing of a mobile Web Service provider for smart phones. *Master thesis, RWTH Aachen, Germany.*, 2004.

[135] Satish Narayana Srirama. Publishing and Discovery of Mobile Web Services in Peer to Peer Networks. In *First International Workshop on Mobile Services and Personalized Environments (MSPE'06)*, pages 15–28, November 16-17, 2006.

[136] Satish Narayana Srirama. MWSMF: A mediation framework for mobile hosts and enterprise on cloud. *International Journal of Pervasive Computing and Communications*, 7(4):316–338, 2011.

[137] Satish Narayana Srirama and Matthias Jarke. Mobile hosts in enterprise service integration. *International Journal of Web Engineering and Technology*, 5(2):187–213, 2009.

[138] Satish Narayana Srirama, Matthias Jarke, and Wolfgang Prinz. Mobile Host: A Feasibility Analysis of Mobile Web Service Provisioning. In *UMICS*, 2006.

[139] Satish Narayana Srirama, Matthias Jarke, and Wolfgang Prinz. Mobile web service provisioning. In *Telecommunications, 2006. AICT-ICIW'06.*

International Conference on Internet and Web Applications and Services/Advanced International Conference on, page 120. IEEE, 2006.

[140] Satish Narayana Srirama, Matthias Jarke, Hongyan Zhu, and Wolfgang Prinz. Scalable mobile web service discovery in peer to peer networks. In *Internet and Web Applications and Services, 2008. ICIW'08. Third International Conference on*, pages 668–674. IEEE, 2008.

[141] Satish Narayana Srirama and Mohan Liyanage. Tcp hole punching approach to address devices in mobile networks. In *2014 2nd International Conference on Future Internet of Things and Cloud (FiCloud)*, pages 90–97. IEEE, 2014.

[142] Satish Narayana Srirama and Carlos Paniagua. Mobile web service provisioning and discovery in android days. In *Proceedings of the 2013 IEEE Second International Conference on Mobile Services*, pages 15–22. IEEE Computer Society, 2013.

[143] Vladimir Stantchev, Ahmed Barnawi, Sarfaraz Ghulam, Johannes Schubert, and Gerrit Tamm. Smart items, fog and cloud computing as enablers of servitization in healthcare. *Sensors & Transducers*, 185(2):121, 2015.

[144] Greg Sterling. Mobile devices now driving 56 percent of traffic to top sites - report - marketing land. `https://marketingland.com/mobile-top-sites-165725`, February 23 2016. (Accessed on 06/14/2018).

[145] Ruoyu Su, Ramachandran Venkatesan, and Cheng Li. An energy-efficient asynchronous wake-up scheme for underwater acoustic sensor networks. *Wireless Communications and Mobile Computing*, 2015.

[146] Xiang Su, Jukka Riekki, Jukka K Nurminen, Johanna Nieminen, and Markus Koskimies. Adding semantics to internet of things. *Concurrency and Computation: Practice and Experience*, 27(8):1844–1860, 2015.

[147] Audie Sumaray and S Kami Makki. A comparison of data serialization formats for optimal efficiency on a mobile platform. In *Proceedings of the 6th international conference on ubiquitous information management and communication*, page 48. ACM, 2012.

[148] Tarik Taleb, Sunny Dutta, Adlen Ksentini, Muddesar Iqbal, and Hannu Flinck. Mobile edge computing potential in making cities smarter. *IEEE Communications Magazine*, 55(3):38–43, 2017.

[149] Nguyen B Truong, Gyu Myoung Lee, and Yacine Ghamri-Doudane. Software defined networking-based vehicular adhoc network with fog computing. In *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, pages 1202–1207. IEEE, 2015.

[150] Wil MP Van der Aalst. The application of petri nets to workflow management. *Journal of circuits, systems, and computers*, 8(01):21–66, 1998.

[151] Charl van der Westhuizen and Marijke Coetzee. A framework for provisioning restful services on mobile devices. In *Adaptive Science and Technology (ICAST), 2013 International Conference on*, pages 1–7. IEEE, 2013.

[152] Aart Van Halteren, Richard Bults, Katarzyna Wac, Dimitri Konstantas, Ing Widya, Nicolay Dokovsky, George Koprinkov, Val Jones, and Rainer Herzog. Mobile patient monitoring: The mobihealth system. *Journal on Information Technology in Healthcare*, 2(5):365–373, 2004.

[153] Rohit Verma and Abhishek Srivastava. A Novel Web Service Directory Framework for Mobile Environments. In *Web Services (ICWS), 2014 IEEE International Conference on*, pages 614–621. IEEE, 2014.

[154] Ovidiu Vermesan, Peter Friess, Patrick Guillemin, Sergio Gusmeroli, Harald Sundmaeker, Alessandro Bassi, Ignacio Soler Jubert, Margaretha Mazura, Mark Harrison, M Eisenhauer, and Others. Internet of things strategic research roadmap. *Internet of Things: Global Technological and Societal Trends*, 1:9–52, 2011.

[155] Leye Wang, Daqing Zhang, and Haoyi Xiong. effSense: energy-efficient and cost-effective data uploading in mobile crowdsensing. In *Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication*, pages 1075–1086. ACM, 2013.

[156] Zhenyu Wu, T Itala, Tingan Tang, Chunhong Zhang, Yang Ji, M Hamalainen, and Yunjie Liu. Gateway as a service: A cloud computing framework for web of things. In *the 19th Int. Conf. on Telecommunications*, pages 1–6. IEEE Press, 2012.

[157] Jiquan Xie, Lilin Dan, Lu Yin, Zhaojie Sun, and Yue Xiao. An energy-optimal scheduling for collaborative execution in mobile cloud computing. In *Computing and Communication (IEMCON), 2015 International Conference and Workshop on*, pages 1–6. IEEE, 2015.

[158] Boyi Xu, Lida Xu, Hongming Cai, Lihong Jiang, Yang Luo, and Yizhi Gu. The design of an m-health monitoring system based on a cloud computing platform. *Enterprise Information Systems*, 11(1):17–36, 2017.

[159] Wang Yong-An, Zhu Bin, and Li Guan-Yu. Gateway-Based Semantic Collaboration Method in SWoT. In *Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2016 International Conference on*, pages 136–141. IEEE, IEEE Press, 2016.

[160] Seungwook Yoon, Kanggu Park, and Euiseok Hwang. Connected electric vehicles for flexible vehicle-to-grid (v2g) services. In *2017 International Conference on Information Networking (ICOIN)*, pages 411–413. IEEE, 2017.

[161] Zhiyong Yu, Daqing Zhang, Zhiwen Yu, and Dingqi Yang. Participant selection for offline event marketing leveraging location-based social networks. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 45(6):853–864, 2015.

[162] Zhuxiu Yuan, Lei Wang, Lei Shu, Takahiro Hara, and Zhenquan Qin. A balanced energy consumption sleep scheduling algorithm in wireless sensor networks. In *Wireless Communications and Mobile Computing Conference (IWCMC), 2011 7th International*, pages 831–835. IEEE, 2011.

[163] Zhang Yun, Gao Ren, and Bian Fuling. A Conceptual Architecture for Advanced Location Based Services in 4G Networks. In *Wireless Communications, Networking and Mobile Computing, 2007. WiCom 2007. International Conference on*, pages 6525–6528. IEEE, 2007.

[164] Weiwen Zhang, Yonggang Wen, and Dapeng Oliver Wu. Energy-efficient scheduling policy for collaborative execution in mobile cloud computing. In *INFOCOM, 2013 Proceedings IEEE*, pages 190–194. IEEE, 2013.

ACKNOWLEDGEMENT

First and foremost, I would like to thank my supervisors Prof. Satish Srirama and Dr. Chii Chang for their valuable advice and support throughout all these years.

I am very thankful to Prof. Marlon Dumas, who approached me on the very first day at this University and gave me much motivational advice and also, Ms. Natali Belinska and staff of International students' service for helping me to come to Estonia. In addition to that, I want to thank all the professors that taught me and guided me during my studies at the University of Tartu and the Tallinn University of Technology.

I am very thankful to my reviewers/opponents Prof. Richard Lomotey, Prof. Jussi Kangasharju, Prof. Raimundas Matulevičius and Dr. Feda AlShahwan for their insightful suggestions and valuable comments.

I would also like to express my gratitude to my friends from Mobile & Cloud Lab, Prof. Eero Vainikko and members of the Distributed Systems group, for all the interesting discussions and support during my studies. And also I want to thank Vitali and Julia Pastsuk for their valuable assistance during my stay here in Estonia.

I also want to thank for the financial support from the European Regional Development Fund through the EXCS, the Estonian Science Foundation grant PUT360 and the Estonian Research Council grant IUT20-55.

Finally, I am grateful for the support from my family: mom, brother, sister, wife, and kids who always with me and giving their utmost support to finish this thesis.

SISUKOKKUVÕTE

Raamistik mobiilsete asjade veebile

Asjade Internet (ingl *Internet of Things*, lüh IoT) tähistab mõistet, kus füüsilised ja virtuaalsed objektid, nagu näiteks autod, kodumasinad, loomad, tööstusmasinad, moodustavad keskkonna, kus need objektid on digitaalselt üheselt identifitseeritavad, võimaldades neil omavahel üle Interneti suhelda. Kontseptuaalses IoT arhitektuuris on peamiseks võrgusuhtluse tehnoloogiaks internetiprotokoll (IP) sõltumata sellest, mis protokollid on kasutusel ülemises rakenduskihis, mistõttu jäävad ühilduvusega seotud küsimused vastuseta. Sellest lähtuvalt on *World Wide Web* (lüh W3C) konsortsium loonud asjade veebi (ingl *Web of Things*, lüh WoT), mis standardiseerib kommunikatsiooniliideseid IoT esemete vahel, eesmärgiga saavutada globaalne, veebitehnoloogiatel põhinev suhtlusprotokolli standard.

Teisalt on mobiilseadmetest nagu nutitelefonid ja tahvelarvutid saanud peamised komponendid mobiilsetes IoT süsteemides. Täpsemalt kätkevad nutitelefonid endas erinevaid riistvara- ja tarkvarapõhiseid andureid ning neil on võimekus vahendada väliste andurite (nt kehaandur) andmeid pilvepõhisesse kesksesse haldussüsteemi. Selleks, et kasutada mobiilseid seadmeid IoT süsteemis anduriandmete allikana või vahendajana saab süsteem majutada mobiilseadmes mobiilseid veebiteenuseid (ingl *Mobile Web Services*, lüh MWS). MWS pakub lisavõimekust mitmete anduripõhiste mobiilsete rakenduste puhul, sealhulgas *mobile crowdsensing*, reaalaja-personaalmeditsiin ja läheduspõhised mobiilsed sotsiaalvõrgustikud.

Sellised targad mobiilsed seadmed paigutuvad mobiilse IoT kategooriasse (lüh MIoT), mis hõlmab füüsiliste, liikuvate objektide (näiteks loomad, droonid, sõidukid või ka inimesed) ühendamist. Arvestades, et olemasolevate mobiilseadmete ja tarkvararaamistike hulk on suur, tõstatub MIoT-s ühilduvuse probleem. Tõhusa MIoT-süsteemide vahelise suhtluse saavutamiseks on võimalik WoT arhitektuuriga kohandada MIoT seadmed mobiilseteks WoT (MWoT) seadmeteks.

Kuigi uusimate mobiilseadmete andmeedastuskiirused ning arvutuslik võimsus on üpris võimekad, ei ole võimalik aku toitel pikaajaliselt jooksutada rakendusi, mis mitmetuumaliste protessorite ja 3G/4G ühenduste võimalusi intensiivselt ära kasutavad. Hulk varasemaid teadustöid on uurinud, kuidas ületada ressursinõudlikkusega seotud probleeme mobiilil majutatavate teenuste vallas. Siiski leidub kitsaskohti, mis vajavad tähelepanu, pidades silmas seadme energiasäästlikkust. Käesolevas doktoritöös esitleb autor kergekaalulist teenusekesket MWoT raamistikku, mis kasutab energiasäästlikke rakenduskihi protokolle.

Tüüpilises IoT süsteemis toimub andmetöötlus ja juhtimine keskses, pilvepõhises haldussüsteemis. Ka MWoT võiks lähtuda sellisest ülesehitusest, kuid arvestades latentsuse probleemi mobiilseadme ja pilve vahelise ühenduse korral, ei suuda klassikaline lähenemine toetada rakendusi, mis nõuavad ülimadalaid reaktsiooniaegu andmete põhjal otsuste tegemiseks. Selliste vajaduste rahuldamiseks

on autor laiendanud kergekaalulist MWoT raamistikku lähisuduarvutuse (ingl *mist computing*) mehhanismiga, mis võimaldab süsteemil osa arvutusülesandeid pilvest MWoT *hostile* jaotada. Võttes arvesse MWoT hosti riistvara piiratust võrreldes pilvega, ei suuda MWoT *host* rahuldada teenuskvaliteedi (ingl *Quality of Service*, lüh QoS) nõudeid olukordades, kus samu riistvarafunktsioone hõlmavate üheaegsete päringute hulk on suur. Selleks tutvustab autor autonoomset ressursiteadlikku teenuste konfigureerimist, mis haldab MWoT hostil majutatavate teenuste kättesaadavust dünaamiliselt muutuva riistvararessursi kättesaadavuse põhjal. Lisaks toetab raamistik ülesannete jaotamist hulga lähisuduarvutust toetavate ressurside vahel.

Autor on uurinud ka MWoT seadmete energiatarbimise vähendamise küsimust, pidades silmas mobiilse internetiühenduse võrdlemist väiksema levialaga tehnoloogitega, nagu Wi-Fi, *Bluetooth Low Energy* (lüh BLE) jt. Ehkki MWoT seadmed võivad ära kasutada avalikke Wi-Fi pääsupunkte, tõstatab see turvalisusega seonduvaid küsimusi. Kui MWoT seadmed aga kasutaksid eelnevalt identifiseeritud ja usaldatud osapoolte pääsupunkte, hoiaks see kokku akuressurssi ja väldiks turvaprobleeme. Hiljutised uurimused IoT valdkonnas asetavad rõhku geograafiliselt kaugetest keskserveritest protsesside jaotamisele IoT seadmete lähedal asuvatele ressurssidele. Sellist lähenemist nimetatakse uduarvutuse ja -võrgu arhitektuuriks ning see pakub MWoT-le alternatiivset võimalust Internetiga ühendumiseks. Kuid udu-võrguteenuste (ingl *Fog Networking service*, lüh FogNets) kasutamine püstitab uusi uurimisküsimusi seoses FogNets võrguteenuste pakkujate paljususe ja heterogeensusega. Viimane seisneb varieeruvates töögraafikutes, töörkoomustes, ootejärjekordades jne. Juhul, kui MWoT seade ei võta selliseid parameetreid arvesse, võib juhtuda, et MWoT seade on sunnitud pidevalt teenuspakkujaid vahetama ning seeläbi energiat raiskama, kuna selgub, et valitud pakkuja töögraafik ei ühildu seadme nõuetega. Taolisi olukordi aitab vältida autori väljapakutud ennetav FogNets MWoT ajaplaneerimisraamistik. Antud raamistik optimeerib FogNets ühenduste loomise graafikut, seades eesmärgiks FogNets teenuspakkujate vahel ümberlülituste vältimise ja sellest tuleneva energia kokkuhoiu.

Lõpetuseks, doktoritöö käsitleb põhjalikult energia kokkuhoidu MWoT jaoks, töö käigus loodud raamistikud on arendatud päris seadmetele ja evalveeritud mitmetes juhtumiuuringutes.

CURRICULUM VITAE

Personal data

Name:	Mohan Liyanage.
Date and Place of Birth:	09.Dec.1971, Sri Lanka.
Citizenship:	Sri Lankan.

Education

2014–2018	Institute of Computer Science, Faculty of Science and Technology, University of Tartu, Doctoral Studies, Specialty: Computer Science.
2013–2014	University of Tartu and Tallinn University of Technology, Master Studies, Specialty: Software Engineering.
2000–2001	Institute of Computer Technology, University of Colombo, Sri Lanka. Postgraduate Diploma in Computer Technology.
1993–1997	Faculty of Science, University of Ruhuna, Sri Lanka. Bachelor of Science.

Employment

2010–2013	Lecturer, Sri Lanka Institute of Information Technology.
2008–2010	ICT Specialist (Cisco Systems), UNDP, Sri Lanka.
2004–2008	Cisco Certified Network Academy Instructor, Sri Lanka Institute of Information Technology.

Scientific work

Main fields of interest:

- Mobile Web service provisioning; Internet of Things; Fog Computing

ELULOOKIRJELDUS

Isikuandmed

Nimi:	Mohan Liyanage.
Sünniaeg ja -koht:	9. detsember, 1971, Sri Lanka.
Kodakondsus:	Sri Lanka.

Haridus

2014–2018	Arvutiteaduse instituut, Tartu Ülikool, doktoriõpe, eriala: informaatika.
2013–2014	Tartu Ülikool ja Tallinna Tehnikaülikool, magistriõpe, eriala: tarkvaratehnika.
2000–2001	Arvutehnika instituut, Colombo Ülikool, Sri Lanka. Kraadiõppe diplom arvutitehnikas.
1993–1997	Ruhuna Ülikool, Sri Lanka, bakalaureuseõpe.

Teenistuskäik

2010–2013	Lektor, Sri Lanka Infotehnoloogia instituut.
2008–2010	IKT Spetsialist (Cisco Systems), UNDP, Sri Lanka.
2004–2008	*Cisco Certified Network Academy Instructor*, Sri Lanka Infotehnoloogia instituut.

Teadustegevus

Peamised uurimisvaldkonnad:

- mobiilsed veebiteenused; asjade Internet; uduarvutus

LIST OF ORIGINAL PUBLICATIONS

1. M. Liyanage, C. Chang, S. N. Srirama: **Adaptive mobile Web server framework for Mist computing in the Internet of Things**, *International Journal of Pervasive Computing and Communications, ISSN: 1742-7371,* 14(3/4):247–267, 2018. DOI: https://doi.org/10.1108/IJPCC-D-18-00023
2. M. Liyanage, C. Chang, S. N. Srirama: **Lightweight Mobile Web Service Provisioning for the Internet of Things Mediation**, *International Journal of UbiComp (IJU), ISSN: 0360-0300,* 8(1):17-34, 2017. AIRCC. DOI: 10.5121/iju.2017.8102
3. M. Liyanage, C. Chang, S. N. Srirama: **Energy-Efficient Mobile Data Acquisition using Opportunistic Internet of Things Gateway Services**, The 9th IEEE International Conference on Internet of Things (iThings2016), Chengdu, Sichuan, China, December 16-19, 2016, pp. 217-222. IEEE.
4. M. Liyanage, C. Chang, S. N. Srirama: **mePaaS: Mobile-Embedded Platform as a Service for Distributing Fog Computing to Edge Nodes**, The 17th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT-16), Guangzhou, China, December 16-18, 2016, pp. 73-80. IEEE. (**Won Best Student Paper Award**)
5. M. Liyanage, C. Chang, S. N. Srirama: **Lightweight Mobile Web Service Provisioning for Sensor Mediation**, 4th International Conference on Mobile Services (MS 2015), New York, USA, June 27 - July 2, 2015, pp. 57-64. IEEE. (**Won Best Paper Award**)

DISSERTATIONES INFORMATICAE PREVIOUSLY PUBLISHED IN DISSERTATIONES MATHEMATICAE UNIVERSITATIS TARTUENSIS