FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING

Ahsan Manzoor

socialAWARE: Mobile Wireless Sensing Network

Master's Thesis
Degree Programme in Computer Science and Engineering
December 2017

Manzoor A. (2017) socialAWARE: Mobile Wireless Sensing Network. University of Oulu, Degree Programme in Computer Science and Engineering. Master's thesis, 48 p.

ABSTRACT

This thesis presents a software tool for Android smartphones called socialAWARE, a mobile wireless sensing network. socialAWARE uses zeroconf networking to discover other mobile devices and their connection information on a local area network. It uses built-in mobile sensors to collect data and transmit it in real time using CoAP's machine-to-machine protocol. SocialAWARE aims at helping users to quickly deploy a wireless sensor network without an emphasis in configuration or technical background.

socialAWARE is implemented as plug-in for AWARE framework [1], which uses diverse protocols to enhance its capabilities. Together, socialAWARE plug-in and AWARE allows for data collection and real time sharing of sensor data between different devices (LAMP server, smartwatch, Android, iOS). After the implementation of the plug-in, the performance of the protocols were evaluated by conducting several experiments. We also compare CoAP with MQTT with respect to their technical performance in terms of latency, throughput and network usage. Based on the experimental results, we discuss the advantages and limitations of the system. Finally, we conclude this thesis by discussing a number of improvements for future iterations of socialAWARE, based on the literature survey and experiment results.

Keywords: smartphones, wireless sensor network, zeroconf, coap, context-awareness, IoT.

TABLE OF CONTENTS

FOREWORD

This thesis was completed at Center for Ubiquitous Computing at the University of Oulu, Finland. I would like to thank my supervisor, Dr. Denzil Ferreira for his continuous support during my studies and thesis. Without his supervision this work would not have happened. I would also like thank Dr Xiang Su and Teemu Leppänen for their supervision. I am also grateful to the Center for Ubiquitous Computing, which provided a platform to learn and improve my skills. I would like to thank my family for their love and support, my friends, Yasir, Hamza and Hamid for their company and fun provided throughout my master's journey. At last, I am also grateful to Finland to provide me the opportunity to pursue my master's degree.

Oulu, 27-11-2017.

Ahsan Manzoor

ABBREVIATIONS

API	Application programming interface
CoAP	Constrained Application Protocol
DNS	Domain Name System
DNS-SD	Domain Name System - Service Discovery
HTTP	Hypertext Transfer Protocol
IoT	Internet of Things
IP	Internet Protocol
Kbps	Kilo-Bits per Second
LAN	Local Area Network
M2M	Machine-to-Machine
MQTT	Message Queue Telemetry Transport
MWSN	Mobile Wireless Sensor Network
mDNS	multicast Domain Name System
NSD	Network Service Discovery
OS	Operating System
QoS	Quality of service
REST	Representational State Transfer
SOAP	Simple Object Access Protocol
UDP	User Datagram Protocol
UI	User Interface
URI	Uniform Resource Identifier
WSDL	Web Service Definition Language
WSN	Wireless Sensor Network

XML	Extensible Markup Language
XMPP	Extensible Messaging and Presence Protocol
WAN	Wide Area Network
Zeroconf	Zero-Configuration Networking

1. INTRODUCTION

The main goal of this Master's Thesis is to build a system that allows the mobile device to recognize each other on a network without manual configuration and exchange valuable information between them. We also investigate the performance of the system and compare them with other protocols used for the machine to machine communication.

The evolution of connected computing has seen a dramatic increase in recent years. Majority of the Internet connections are devices used directly by the humans such as mobile phones and computer [2]. In near future, most of the devices and objects can be connected and they can exchange information by themselves and number of devices communicating will be much larger than the number of people [3]. We are entering a new era of ubiquity where machines will be communicating with other machines on the behalf of the people [2]. The main challenge will be to connect them in a meaningful way to deliver the information.

Mobile phones are not only used as communication device but it comes with number of sensors. In recent times, these sensors were mainly used to enhance the user experience. But with the more availability of mobile phone and increase number of sensors these mobile phone has been used for research purposes. It has given rise to a new area of research called mobile phone sensing. Today's mobile phone continue to provide more computation, memory and sensing capabilities but they still remain a resource-constrained device. Therefore, we provide an application for deployment of a mobile sensing network with zero-configuration capabilities and transfer of the data using constrained protocol.

The application developed in this thesis is an extension to AWARE framework[1]. The application will use zero-configuration to register the device as a web service on the local network. After the discovery of device, the host mobile will offer its different senors as a service to the connected mobile and RESTful API will expose those services to access and manipulate the data. In the end, machine-to-machine protocol will be used to transfer the sensor values between the devices.

1.1. Web Services

According to World Wide Web Consortium (W3C), web service is a software system or a function that is identified by the URI, accessed through the XML encoded messages and transmitted using Internet protocols [4, 5]. Web services can be divided into three main components - SOAP that enables communication, WSDL that provides service description and UDDI for service discovery [6]. Web services provide a framework for application-application interaction, without the knowledge of each other systems. It allows different application from various sources to communicate to each other and share data and services among themselves. It makes the application, platform and technology independent e.g. Java web service can talk with Perl and vice versa.

1.2. Representation State Transfer

Representation State Transfer (REST) is a software architecture style and approach to communication often used in web services [7]. REST is easier to use and consume less bandwidth compared to SOAP which makes it more suitable for Internet usage. In order to be considered RESTful, the application should have following characteristics [8]:

- Functionality divided into distributed resources
- Every resource is uniquely addressable using specific command (GET,POST,PUT or DELETE)
- Protocol is stateless, client / server, layered and support cache.

RESTful API is application program interface (API) that exposes the web services by using HTTP request of GET, PUT POST and DELETE for retrieving, creation, mutation and deletion of data. RESTful API makes an application modulator, thus providing developer with a lot of flexibility and creating programmable web.

1.3. Internet of Things

Internet of Things (IoT) can be defined 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" [9]. Some researchers have also defined IoT in terms to internet related aspects, such as internet protocols and network technologies. The vision of IoT is the integration of billions of smart objects for interaction, that will lead to network devices communicating and exchanging data with humans and between other devices [10]. In order to achieve ubiquitous and autonomous connectivity between IoT devices, researchers have come up with new Machine-to-Machine (M2M) communication paradigm. These M2M communications are the enabling technologies for the practical realization of IoT. However, IoT still lacks worldwide accepted and standard protocols [11].

In IoT, most of the devices used are low power short range wireless devices with a specific purpose e.g. sensors. Because of this, the current internet communication protocols cannot be used and new M2M protocols are designed to cater the requirements of these IoT devices. These protocols have small payload enabling faster communications and uses less power for longer battery timings. Some of the widely used protocols for machine-to-machine communication are 6LowPAN, multicast-DNS, DNS-Service Discovery, CoAP, MQTT etc.

1.4. Machine-to-Machine Protocols

1.4.1. Constrained Application Protocol (CoAP)

CoAP is a REST based transfer protocol developed to use with constrained networks [12]. It includes several HTTP functionalities that have been redesigned to suit the M2M and IoT applications [13]. CoAP uses a server / client model like HTTP and can act as both in M2M interaction. Some of the other key features of CoAP are:

- HTTP like methods: GET, POST, PUT and DELETE.
- User Datagram Protocol (UDP) allows multicast and has lower overhead.
- 4 message types: Confirmable, Non-confirmable, Acknowledgement and Reset.
- Block-wise transfers to support large files [14].
- Resource discovery allows the user to discover the resources offered by server [15].
- Resource observation allows the user to subscribe to resource and notified when it changes [16].
- Universal Resource Identifiers (URI) based resource representation [15].
- 3 Response codes: 2.xx (success), 4.xx (Client error) and 5.xx (Server error).

CoAP is a platform independent approach and can be easily integrated with the current web technologies. As CoAP is still under development, there exist some limitation that needs to be addressed. Brachmann et al. have identified two problems with CoAP in his work [17]. The first is to provide end-to-end secure connection and second to provide secure multicast communication.

CoAP have many implementations in different programming languages with different features. Libcoap [18] and Californium [19] are the most mature and easy-to use implementations of the CoAP [20].

1.4.2. Bonjour / Network Service Discovery

Bonjour is the implementation of zero configuration networking (zeroconf) [21] by Apple. Inc. [22] while Network Service Discovery is developed by Google [23]. They discover devices and services on the local area network using IP protocols, without requiring the user to configure them manually. Both of them uses the same technology, a combination of mDNS with DNS service discovery . Multicast DNS [24] sends UDP request using port 5353 to all devices connected to the network and other services reply with their IP address and name. They both provide the following features [22, 25]:

- Allocating of IP addresses through IPv4 Link-Local Addressing (IPv4LL)
- Naming of the host through mDNS.

- Service discovery through mDNS / DNS-SD.

Zeroconf helps the user to see what services are available on the network, and can choose from the list. On the other hand, it helps the developers to build the applications that can detect other services over the network, and can communicate without user intervention.

Bonjour was introduced in 2002 as part of Mac OS X but now it works on many other operating systems like Linux, Windows, and Solaris. It is widely used by Apple software to share resources or locate servers. Bonjour provides easy-to-use programming interface with implementations in multiple languages such as Python, Ruby, and Cocoa.

NSD was made available in 2012 starting from Android 4.1 and support multiple devices like printer, web-cams, and mobile. It is mostly used in the development of the Android application. NSD is also compatible with bonjour running on any other system. It provides all the functionalities of bonjour except sharing of the meta-data about the service.

1.4.3. Message Queue Telemetry Transport (MQTT)

MQTT is a messaging protocol that uses publish / subscribe pattern to work with constrained devices and unreliable networks [26]. It is designed to work with minimum device resource and network bandwidth while ensuring reliable delivery. These features of MQTT make it ideal for the IoT and mobile devices. MQTT consist of three components: publisher, subscriber and broker. Some of the main features of MQTT are [27]:

- Multiple communication paradigm: one-to-one, one-to-many, many-to-many
- Three QoS levels.
- Last Will and Testament" to signal device disconnection.
- Security by checking authorization of subscriber and publisher [28].
- Persistence Message support

1.5. AWARE

AWARE is an Android framework that can be used for capturing, inferring and generating mobile context information [29]. AWARE can be used by developers to create context aware applications or researchers can run their studies that involve mobile devices or smart phone users can use the client to record their own data. AWARE captures hardware, software, and human-based data. AWARE allows the users to create plugins that can record the data from multiple sensors, analyze them and transform them into information the user can easily understand.

AWARE is currently available as a regular mobile application on Google Android and Apple iOS. It is an open source platform that enables users to save time and effort

to develop context aware application. AWARE shifts the focus from software development to data analysis and can be used from small to large scale deployments [1].

1.6. Thesis Structure

The organization of the thesis is as following: Chapter 2 provides a comprehensive look at the background of Internet of Things protocols and wireless sensor networks. Chapter 3 presents the detailed information about the system design, architecture and implementation of the application. Chapter 4 contains a detailed description of the evaluation and comparison of the implemented protocols with other IoT protocol. Chapter 5 provides results and analysis of the evaluation performed in the previous chapter. In the last, Chapter 6 provides a conclusion based on the findings. It also provides limitation and summary of the work done.

2. RELATED WORK

This section presents a quick overview of the related applications in this thesis. As the thesis focuses on CoAP, Zero-Configuration Networking and Wireless Sensor Network, this section is categorized in the similar format.

2.1. Constrain Application Protocol (CoAP)

The research on the CoAP has mainly focused on the performance of the protocol, its user interaction and auto-configuration of the network [30]. Amaran et al. [31] evaluated the performance of CoAP with MQTT and proved that MQTT perform 30 % better. Colitti et al. in their work have provided an evaluation of the CoAP compared to HTTP [13]. They came to the conclusion that CoAP has a compact packet overhead allowing CoAP server to spend less time and making it energy efficient and lower response time compared to HTTP. They also verified that CoAP is not sensitive to increase of client request interval time. Another work [32] verifies that the CoAP uses less energy when compared to HTTP. Bormann et al. [33] in their article explain the inter-working of the CoAP with HTTP. As both uses REST architecture, it enables this communication through proxies, which behave like a server to a client and play a client toward another server. The illustration of the inter-working can be seen in Figure 1. Kovatsch et al. [34] implemented CoAP for Contiki [35] operating system and evaluated the performance of the protocol. The authors demonstrate that the performance of CoAP over radio duty-cycle decreases the energy consumption at the cost of higher latency.

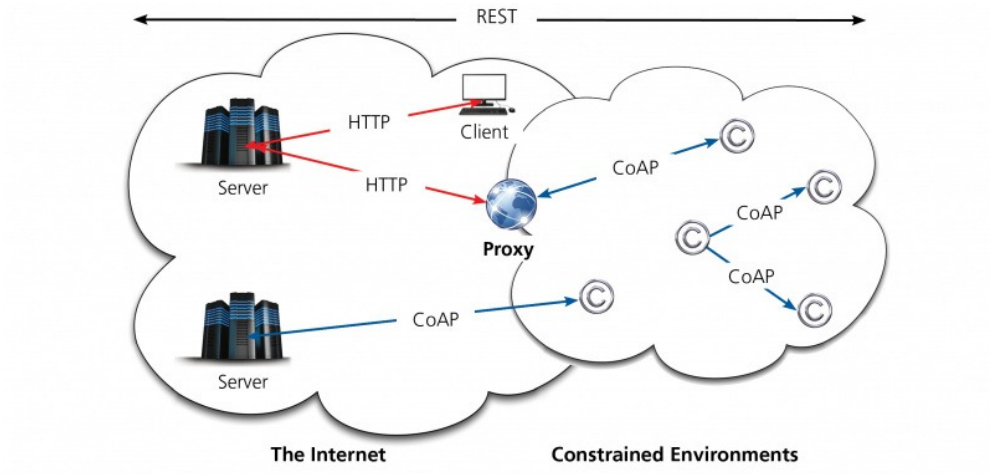

Figure 1. CoAP and HTTP work together [33]

Kovatsch [36] in his work has focused on the two main challenges faced by the Internet of Things (IoT): a scalable application layer for interoperability and reliable programming model. He proposed the solution to these challenges with the use of CoAP, allowing for interoperability and a common programming model along with other benefits of CoAP for the Internet of Things. Lerche et al. [37] discuss the re-

sults of the first formal interoperability, organized by the European Standards Institute (ETSI) in March 2012. At the meeting, 16 clients and 16 server implementations were tested against each other in 27 different test cases and the result implies a high interoperability between implementations.

Researchers have come up with different ways to interact with the devices in the IoT and CoAP being the transfer protocol for IoT [15] lacks user interaction and presentation. Kovatsch developed Copper [38] (Figure 2), a web browser integrated with CoAP protocol suite. Copper allows the user to observe and control the devices in a constrained environment and help in network management. Schor et al. have proposed a web service based solution to integrate and manage these devices [39].

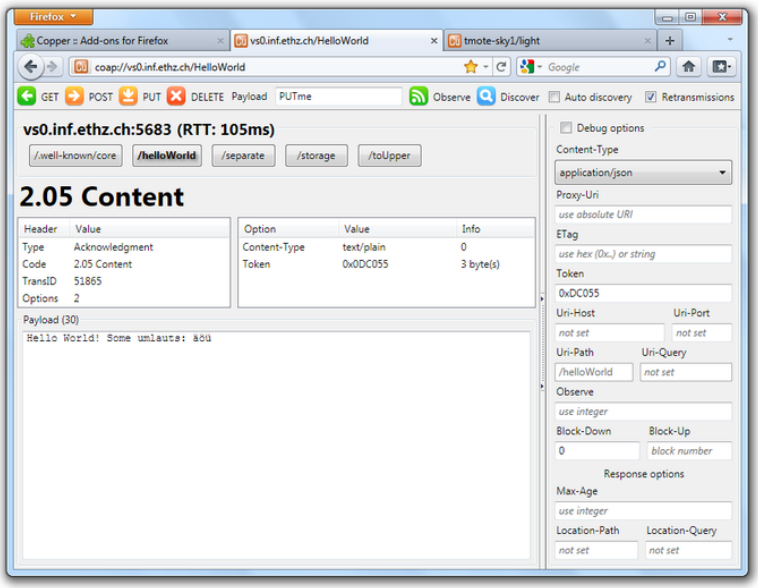

Figure 2. Copper - Firefox plug-in for CoAP [38]

CoAP has been recently used for smart grids and building automation applications [30]. Rahman et al. have proposed another use of the CoAP, group communication by using the multicast feature of the protocol [40]. This group communication can help in building management application as they allow one device to communicate with several others.

The authors of [41] came up with new approach of binding SOAP to the CoAP protocol. They were able to reduce round trip timing up to 43 % compared to widespread TCP over HTTP binding. This approach can be used to deploy SOAP Web services in a constrained environment.

2.2. Zero-Configuration networking

As smart devices have become more popular, research has focused on ways to improve the discovery of these devices without explicit human administration. Zeroconf has multiple implementations with Bonjour [22] being most known and Avahi [42] for Linux systems. Gwizdz in his master's thesis demonstrate Zeroconf communica-

tion between Apple iOS and Android devices [43]. Both the devices automatically discover each other using mDNS over Wi-Fi network and uses HTTP for file transfer. Klauck et al. [44] presents a case study of using Bonjour in constrained devices and operating systems. They showed that existing standards can be used economically on constrained devices and these devices are aware of the topology changes. Siddiqui et al. [45] in their work compared popular implementations of Zeroconf , Avahi and Mono.Zeroconf. Bardin et al. [46] proposed service oriented component framework for the discovery of the nodes with the help of residential gateway. The authors of [47, 48] have also implemented and tested mDNS for Contiki and verifies that they are suitable for constrained devices. Another implementation of Zeroconf for constrained devices can be found in [49]. It implements both mDNS and DNS-SD for Arduino board and also discover services registered by other nodes on the network.

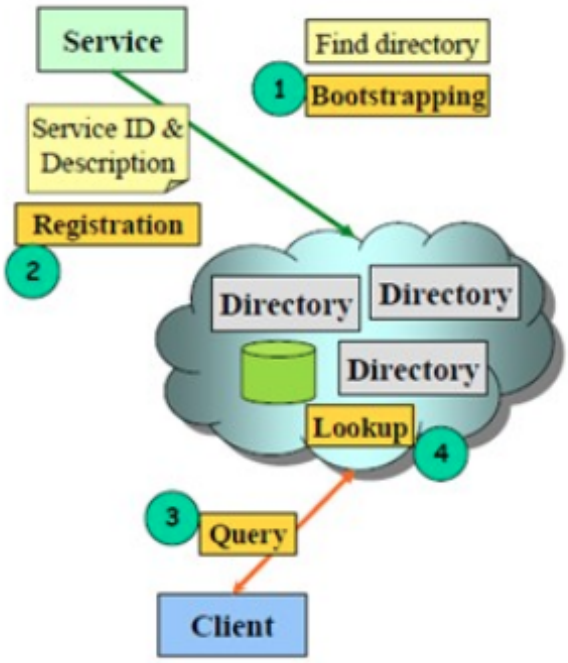

Figure 3. Service Discovery Architecture [50]

The most visible part of Zeroconf is service discovery. Figure 3 illustrates the steps involved in the service discovery. There have been a lot of research on integrating protocol and services from traditional network into constrained environments and it appears not much work has been done in service discovery [47]. Yazar et al. in their work used web service approach to integrate sensor networks with existing infrastructure but it only works with the applications within the scope of web services [51]. The authors of [52] presents the deployment of service discovery protocol in a sensor network using IP and Bluetooth protocols. Their experiments can verify that service discovery of nodes in sensor network is feasible.

Zeroconf uses multicast to remove configurations in service discovery, however this makes it difficult for Zeroconf services to reach beyond local link. Wide area Bonjour [53] is Apples solution for global service discovery. Wide area Bonjour uses unicast

DNS for discovery and requires to maintain a DNS server. In summary, it requires configuration. Cisco also provided a solution for the wide area service discovery based on distributed and hierarchical Bonjour service learning and distribution architecture [54]. Figure 4 displays the topology of the proposed solution by cisco. Lee et al. [55] presents a z2z toolkit based on Bonjour and OpenDHT that extends the reach of existing Zeroconf applications beyond local link and requires no configuration. Their toolkit have implementation issues and it does not work behind a Network Address Translation (NAT) gateway. Authors have also raised privacy concerns when using this toolkit in large networks.

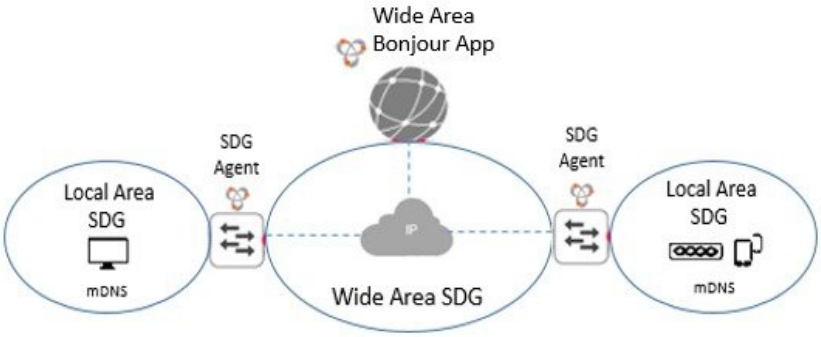

Figure 4. Cisco - Wide Area Bonjour Topology [54]

2.3. Wireless Sensor Network

Wireless Sensor Network (WSN) consist of huge number of nodes with sensing, computation and wireless communication capabilities [56]. On the other hand, Mobile Wireless Sensor Network (MWSN) consist of group of nodes with the ability of move and interact with the environment [57]. They can relocate and organize themselves in the network and can communicate with other nodes within their range.

The current trend in WSN is to move away from the closed standards and enable IP based sensor networking [58].In this way, sensors are interconnected and can participate to the Internet of Things. This trend is also confirmed by ZigBee alliance and they choose IP for their smart energy 2.0 profile [59]. The introduction IPv6 over Low power Wireless Personal Area Network (6LoWPAN) [60] enables the use of IP in constrained environments thus allowing to form IoT. One of the main benefits of using IP is to enable web services on the sensor nodes. Various reseacrers have been proposed REST framework for WSN. Dawson et al. [61] has presented their design of RESTful web services that allows instruments and devices to directly publish their data. Kovatsch et al. [62] have also proposed a REST framework for the home automation. In [63], the authors have presented a real world implementation of RESTful WSN. They deployed the network in various university buildings and developed services for the university community.

Existing WSN technologies have high heterogeneity that has enforced the researcher to find solution to integrate non-IP WSN to the internet. Aberer et al. [64] has presented their Global Sensor Network, an open source framework that supports fast deployment, flexible integration and discovery of sensor networks using a custom middleware and 6LoWPAN. Another approach to IP based WSN is presented in [58]. In their approach they have used a middleware for the connection between IPv6 and legacy networks to create a heterogeneous WSN. Figure 5 shows their proposed architecture.

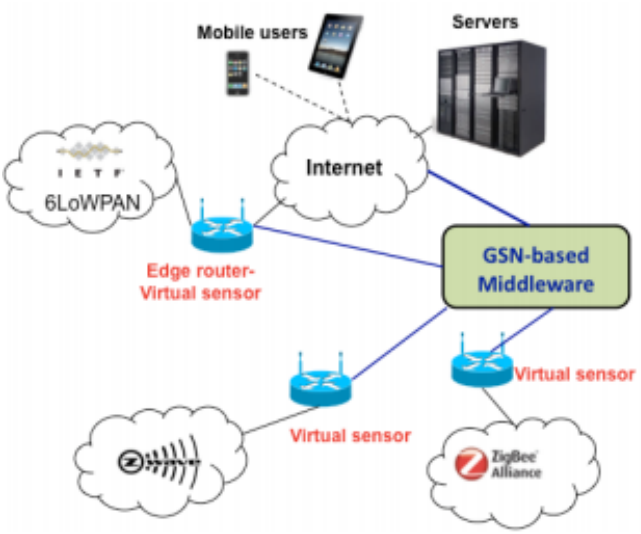

Figure 5. Proposed architecture for heterogeneous WSN through internet [58]

Mobile WSN has several advantages over static WSN such as improved coverage, energy efficiency and enhanced tracking [65]. Liu et al. [66] have shown that mobile sensor nodes has an improved coverage and they also used game theoretic approach to prove enhanced tracking in mobile WSN. The researchers [67] have calculated channel capacity for mobile WSN and proved it to be 3-5 times better than static WSN. On the other hand, a survey on mobile WSN [68] has listed some its challenges. Due to the mobility, mobile WSN have a dynamic topology that causes problems in determining the position of the node. Dynamic topology also have unreliable communication links thus minimum Quality of Service becomes a challenge.

WSN and mobile WSN plays a key role in several applications like military, healthcare, environment monitoring and smart metering. Li at al. [69] have shown the use of WSN as detection and surveillance in military applications. [70] shows demonstration of multi vehicle tracking using sensor network. Sensor networks can be used to monitor environment and building structures thus can be useful to inform or prevent any disaster in advance [71, 72]. New technologies and protocols have further enhanced the capabilities of sensor networks, allowing the researchers to make more applications.

2.4. Overview

It seems that there are no applications or projects strictly like ours. Most of the projects implement zero-configuration on an intranet to configure devices or share services. These discovery protocols are mostly implemented on computer or tiny sensor nodes. As mentioned above, CoAP is mostly used as transfer protocol in constrained computer devices.

In our work we will focus on implementation of socialAWARE, mobile wireless sensing network. This application will provide an easy setup of sensor network using mobile devices. We will use Network Service Discovery, an Android implementation of zero-configuration to discover devices over the network. socialAWARE will use CoAP as a communication protocol to transfer real-time sensor data between the devices.

3. DESIGN AND IMPLEMENTATION

This chapter describes the design and implementation of the socialAWARE, a mobile wireless sensing network software tool. It contains the detail design of the system which is then used to list the system requirements. It also includes the system architecture and the description about the implementation. Lastly, we showcase the user interface of socialAWARE.

3.1. System Design

socialAWARE makes use of multiple existing systems and protocols: AWARE, CoAP, zeroconf. We implemented the zero-configuration for mobile nodes discovery and the use of CoAP as transfer protocol and discovery of the resources. AWARE Android mobile application allows the user to control the sensors and plug-ins. AWARE also allow the researchers and developers to create their own application using the AWARE API.

In this thesis, we have developed a plug-in for the AWARE client that will further enhance the client capabilities. The prime objective of the plug-in is to enable a system where user can deploy a sensor network using the mobile devices and can monitor the data remotely, in real-time. The first part of the thesis focuses on the discovery of the mobile devices using the developed AWARE plug-in. The users should be able to connect the mobile devices without any configuration. The second part focus on using of a protocol for resource discovery which in this scenario can be sensors or plugins' value. That last part focuses on using the lightweight M2M protocol for fast transfer of the data. Figure 6 shows the model of the system

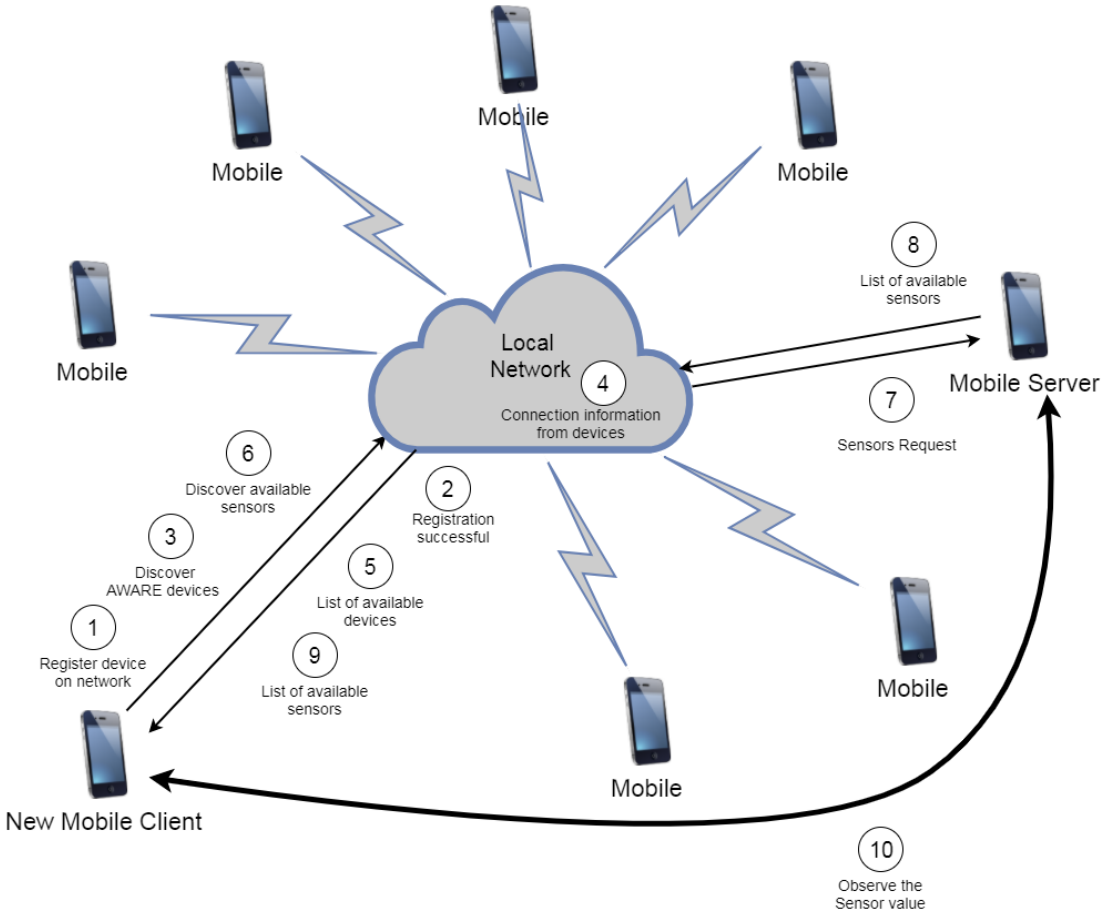

Figure 6. Model diagram of the system

3.2. Requirements

In this section, we provide the system requirement deduced from the design of the system. The following requirements were set for the system.

- Any connected mobile device should act as client / server.
- Mobile device should be able to connect to another device without any configuration.
- Multiple client devices should be able to connect to one device at any given time (many-to-one).
- The mobile device should be able to register or announce itself as a service on the connected local network.
- The mobile device should be able to discover other devices using the AWARE plugin over the connected local network.
- The mobile device should be able to resolve and list the connection information all the other devices with their connection information.

- The mobile device should keep the list of available sensors on its device and share it with other devices.
- Upon user's request, the client device should be able to discover the sensors offered by the connected devices.
- Upon user's request, the client device should be able to observe the sensor values on connected device.

3.3. System Architecture

In this section, we explain the system architecture of the mobile sensing application. System architecture was constructed on basis of system requirement. As this system consist of multiple subsystems i.e. device discovery and constrained application protocol, both together form the entire system. Figure 7 shows the components of the main combined architecture.

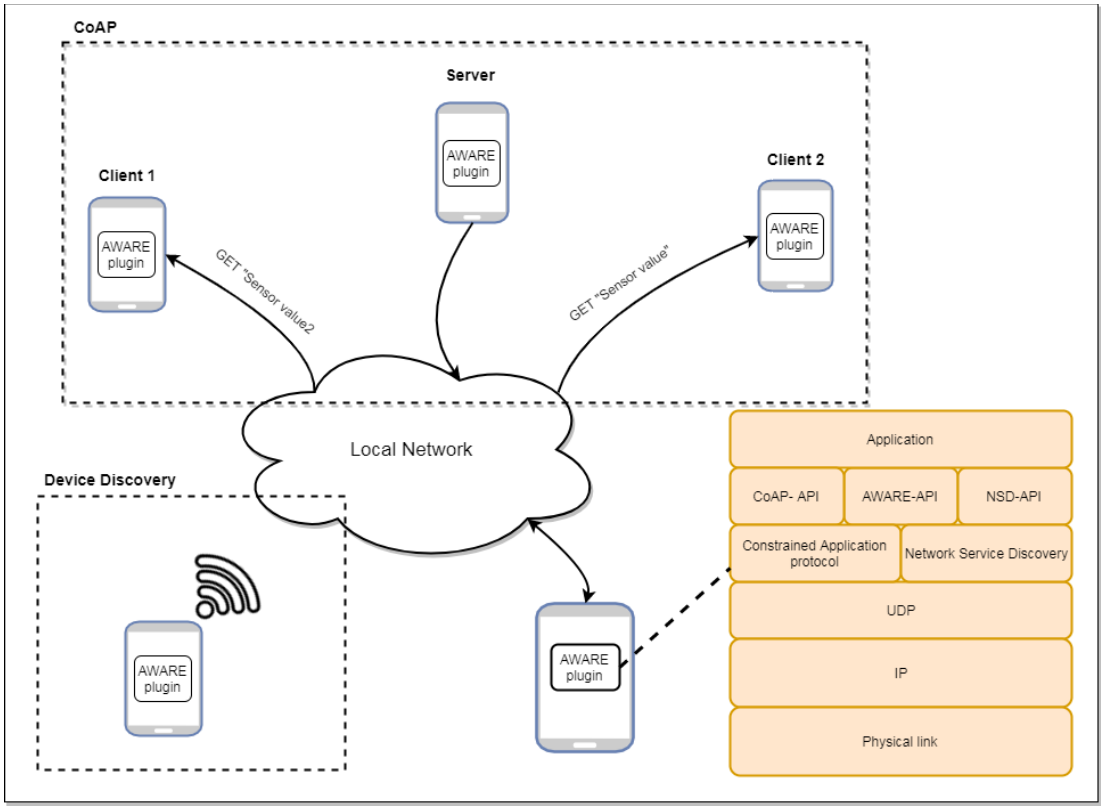

Figure 7. AWARE: mobile sensing network architecture

The application will be built on AWARE framework, taking advantage of existing AWARE APIs. As the mobile user joins the network, it will search for other available devices with AWARE plugin. NSD will be used to discover the devices using the multi-cast DNS messages. These messages will be broad casted over an IP network and will use UDP as transport layer protocol. After successful discovery, AWARE

API will be used to offer the available sensors as resources to the connected client. Sensor data from the mobile will be sent using the CoAP over the same network and using same protocols. Mobile devices can act as a client or server, depending upon the situation. Multiple devices can connect to a single device and can observe the sensor values simultaneously.

3.3.1. Subsystem Architecture

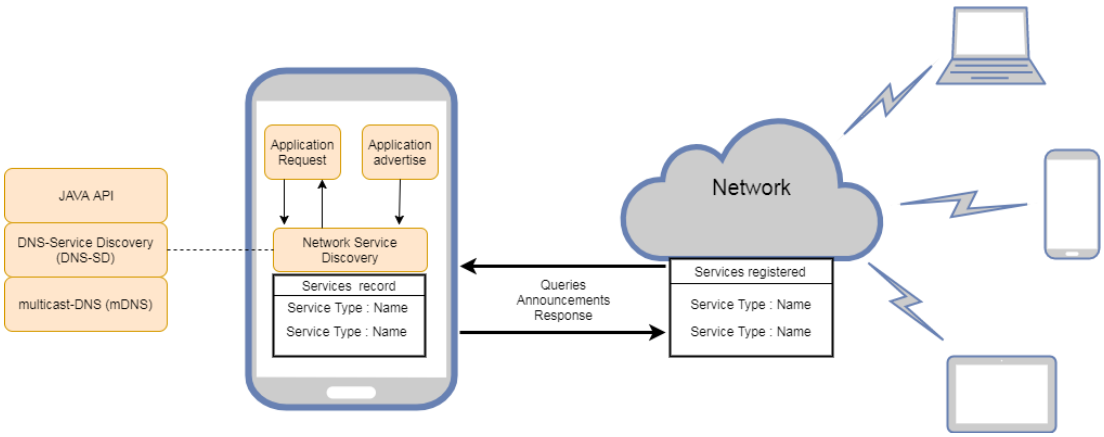

Figure 8. Device discovery subsystem architecture

When an application starts a new service, it advertises the service to a multi-cast address with service type and name. The advertisement of the service is done by the multi-cast-DNS and it follows a naming convention i.e. “<Name>._<protocol>._<transport layer>._<Domain>”. After the registration, DNS-Service discovery is used to find all the particular type of service on the network e.g. “_http._tcp”. Multicast-DNS is then used to return the name of the services found and add them to the cached list on the device. When an application wants to use a service, multicast-DNS resolves the chosen service name to an IP address and port.

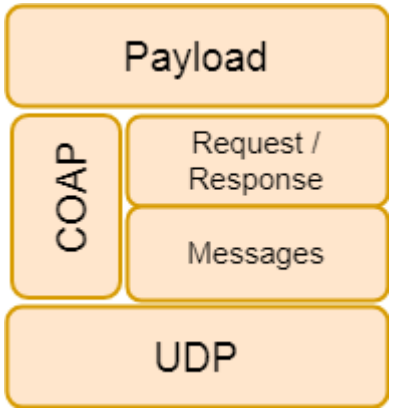

Figure 9. CoAP subsystem architecture

CoAP architecture is very simple and based on REST. Request / Response sub layer is used for communication and RESTful interaction using GET, PUT, POST and DELETE commands. URI and media types are also defined in this layer. The second sub-layer is responsible for reliability and duplication of the messages. In the last, CoAP uses UDP as transport protocol.

3.4. Implementation

This part includes the implementation process in detail. In addition, it provides the sequence diagram to show the object interaction according to the time sequence.

3.4.1. Device Discovery

As the plug-in was created for AWARE Android client, device discovery was implemented using Network Service Discovery. NSD implements mDNS and DNS-SD that allows the application to identify the devices on the local area network. As the user starts the plug-in, it will broadcast its name and connection information to the connected local network and it will also search for the information from other devices doing the similar work. Figure 10 shows the sequence diagram for device discovery in AWARE plug-in.

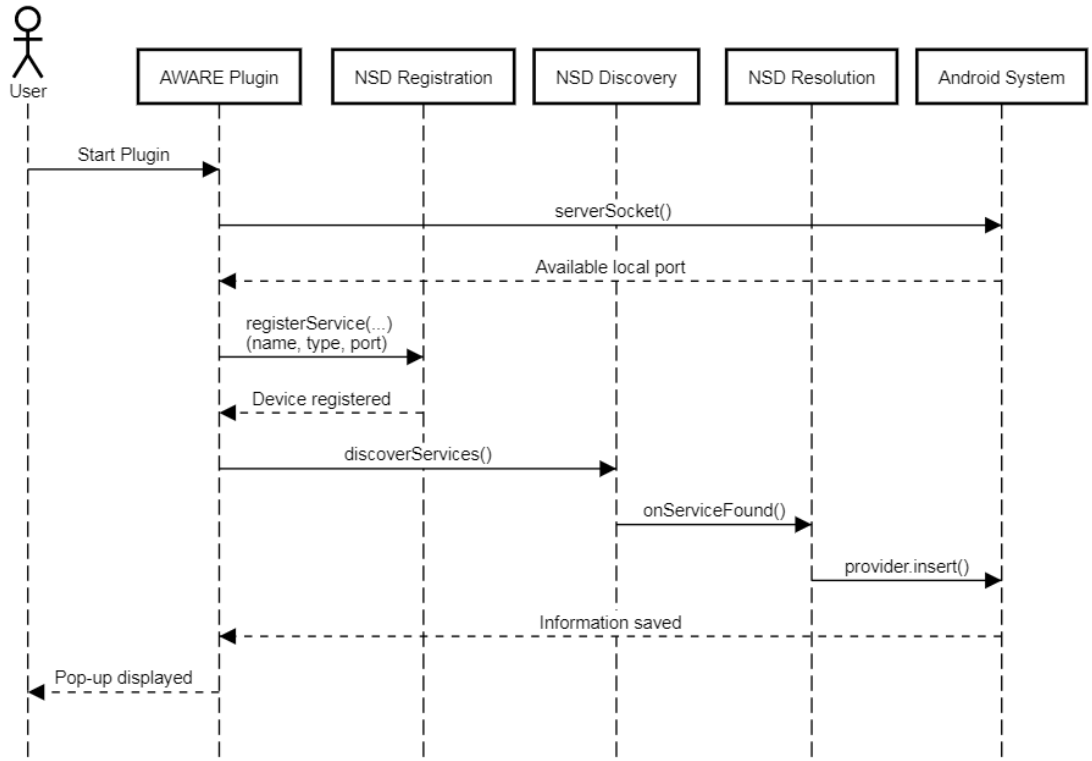

Figure 10. Sequence diagram for NSD device discovery

The first step towards the implementation of device discovery requires the application to register the device as service on the local network. In order to register the service for the plug-in, NSD requires a unique name, service type, and port number. The name of service should be unique for every device, so it was decided to use Android ID concatenated with "AWARE" string. Android ID has a different value for each application and each device. HTTP protocol was selected as the service type because the plug-in uses CoAP as the transport protocol which is much like HTTP. The last parameter was to select the port number on which the plug-in will register its services. To avoid the conflict, the dynamic port number was assigned. Every time, server socket searches for the device's next available port and NSD uses that port number to register its services. The code used to register the service look like the following:

```
public void registerService(String ID, int local_port) {
        NsdServiceInfo serviceInfo  = new NsdServiceInfo();
        serviceInfo.setServiceName("AWARE_" + ID);
        serviceInfo.setServiceType("_http._tcp.");
        serviceInfo.setPort(local_port);
        NSDMANAGER.registerService(serviceInfo,
        NsdManager.PROTOCOL_DNS_SD, REGISTRATIONLISTENER);
    }
```

After the device registration, the plug-in needs to search for the similar devices registered on the connected local network. Device discovery was divided into multiple steps. The first step was to initialize a Discovery listener which starts searching names of all the devices registered on the network. This discovery listener only searches for devices using the HTTP service type, as their might be other services registered on the network.

```
public void discoverServices() {
        NSDMANAGER.discoverServices("_http._tcp.",
        NsdManager.PROTOCOL_DNS_SD, DISCOVERYLISTENER);
    }
```

Once the device service is found, the listener resolves the name and using the "AWARE" tag, filters out the required devices. In the last, discovery listener sends the service information to the NSD manager which then further resolves the connection information.

In the last step, NSD manager resolves the IP address and port number of the service by looking at the multicast DNS packet. This information is then stored locally on the mobile device using the Android content provider. The stored information is later used to create a network connection between the mobile devices.

As device discovery is an expensive operation, it needs to be stopped when the work is completed. It is also important to unregister the device from the connected network as it prevent active devices to connect to the inactive ones. The developed plug-in automatically unregisters its services and stop device discovery operation when the user stops the application.

3.4.2. CoAP Resource Registration and Discovery

After the mobile device has been discovered and connection has been made between them, the next step was to discover the resources offered by the connected device. CoAP, which was selected as the transfer protocol also supports resource discovery. In CoAP, a "GET" packet request is sent to the device acting as the server and it replies with the list of their resources along with their media types. In order to discover the resources, the server device must first register these resources and maintain a list at "/.well-known/core", which can later be accessed by any client device. Figure 11 shows the sequence diagram for the registration and discovery of the resources.

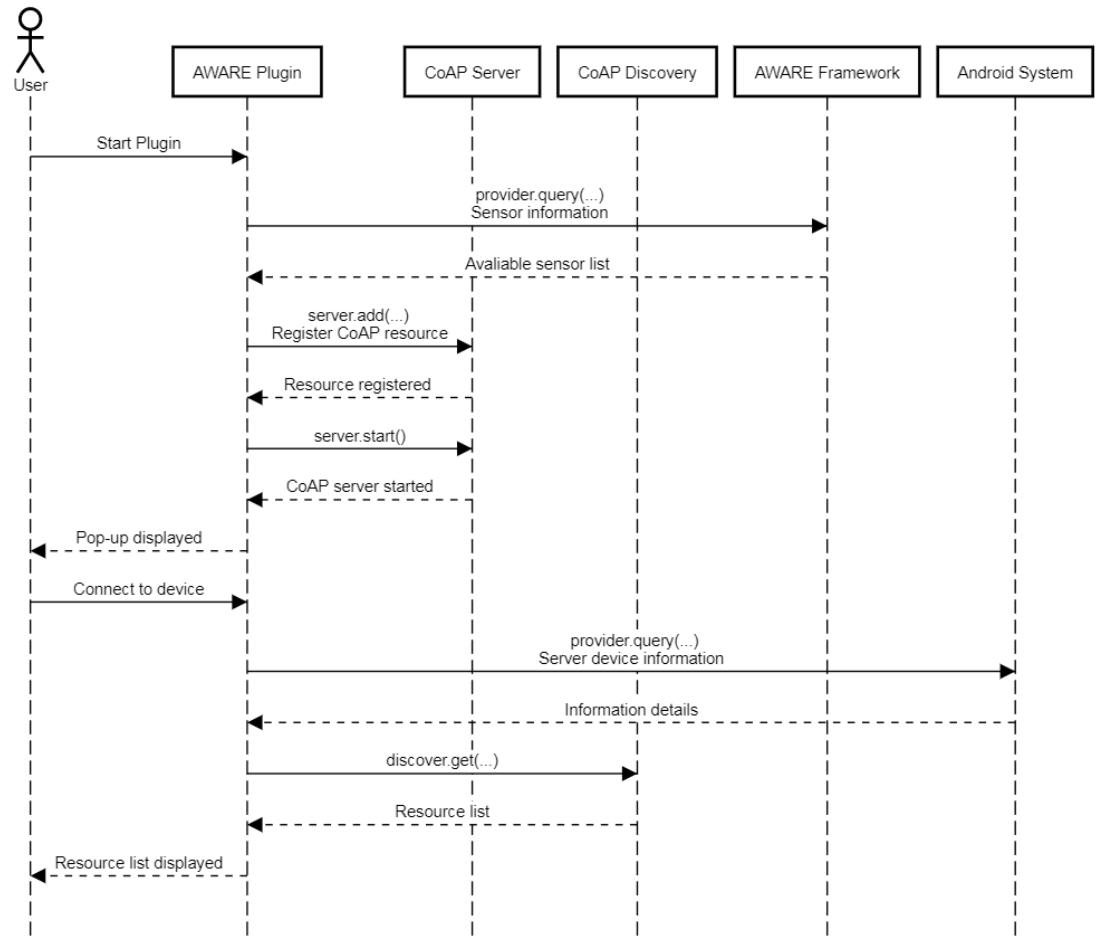

Figure 11. Sequence diagram for CoAP Resource Registration and Discovery

The implementation of the CoAP server is vital for the device to offer its resources. Our plug-in should search for the activated sensors on the server device and offer them as resources to the client. AWARE platform maintains a content provider which records the information of all the activated sensors. When the user starts the plugin, we filter out the available sensors and register them as resources. CoAP server registers a resource when provided with a unique name. We used the name of the sensor as resource name. After the resources are added, the CoAP server is started. In order to add a new resource, the user needs to restart the CoAP server and register all the

resources again. We registered these resources as observable so it can notify the client whenever the data from sensor changes.

```
public universalResource(String name, Context context, boolean vi
        this.linkName= name;
        this.mContext = context;
        setName(linkName);
        setObservable(visible);
        setObserveType(CoAP.Type.CON);;
        getAttributes().setTitle(linkName);
        getAttributes().setObservable();
    }
```

Once the resource was registered, we needed to implement CoAP discovery for the plug-in. CoAP discovery on the client device uses the connection information from the NSD and sends a "GET" request to the server. The response from the server is then listed to the user and URI link to that resource is stored temporarily.

As the plug-in should be power efficient, CoAP discovery is only active when the user connects to a specific device. It also automatically stops the CoAP server when the plug-in is stopped

3.4.3. Transfer protocol

The last step in the application was to implement RESTful services for the provided resources. CoAP provides HTTP like methods for RESTful communication. CoAP "GET" method was implemented to request data from the server and "PUT" method for the mutation of the data. These methods should be predefined when the CoAP server registers the resource.

AWARE framework uses broadcasts to quickly update applications with the sensor values. As our plugin gives an option to the user to observe a specific resource, we had to implement a broadcast receiver in the CoAP server to receive the updated value and change it accordingly. Once the value is changed the CoAP server is then notified which informs the client with the updated value without user request. The following code connects to a CoAP server and observes a resource.

```
CoapClient client = new CoapClient(URL);
client.observe(new CoapHandler() {
        @Override
        public void onLoad(final CoapResponse response) {
          response_code = response.getCode();
          response_value =response.getResponseText();
            }
    });
```

The combined sequence diagram of the whole system is shown in the figure 12.

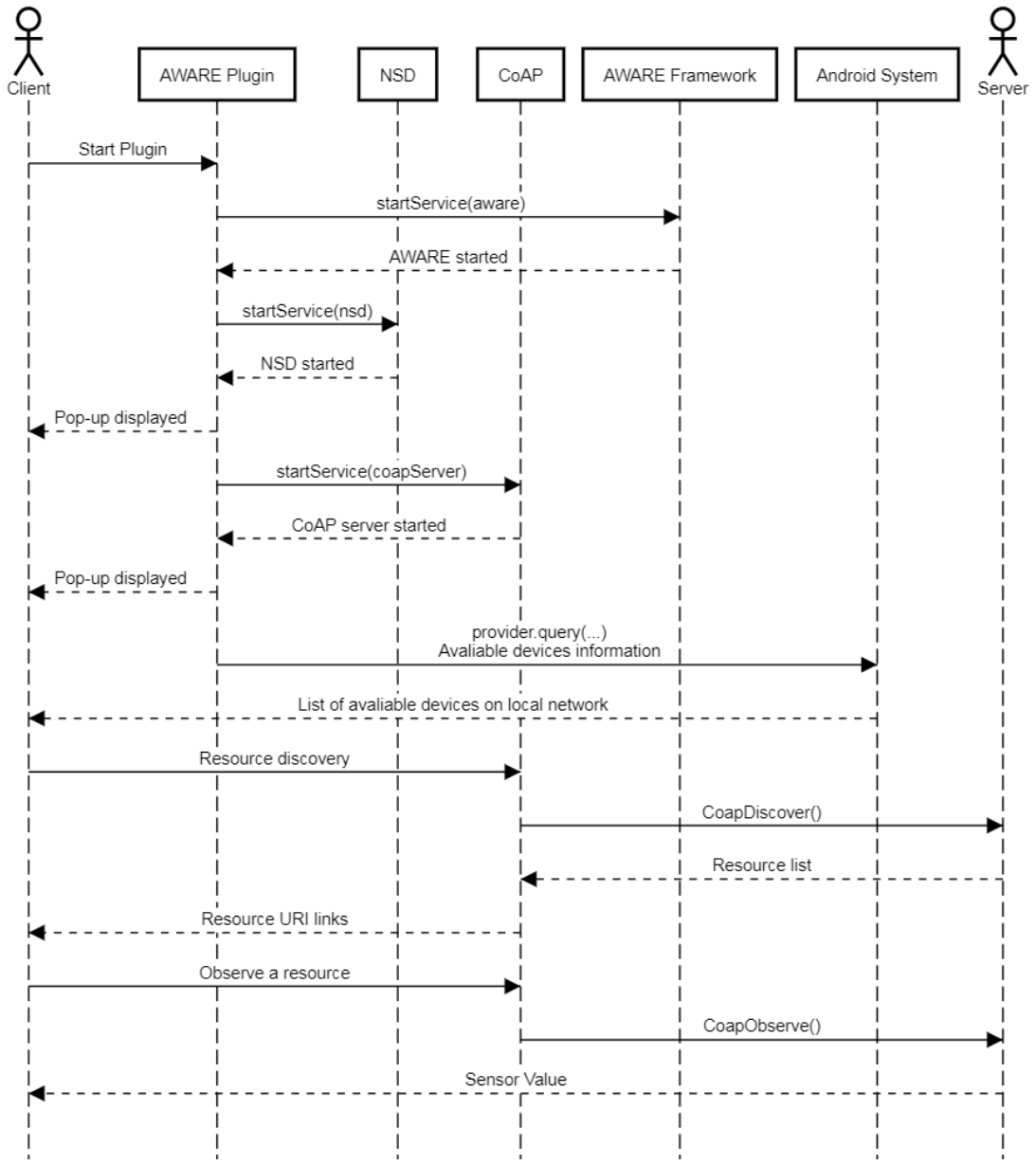

Figure 12. Combined sequence diagram for AWARE plugin

3.5. Class Diagram

A class diagram shows the structure of a system by showing set of classes, attributes, methods and their relationships [73]. The class diagram for the socialAWARE plug-in is shown in Appendix 1.

3.6. User Interface

When creating the user interface of the application, the aim was to keep the interface as simple and clear as possible. Another aim of the interface design was to keep it as close as possible to the AWARE application and device design, so it does not stand out.

After installing the designed plug-in the AWARE application launches the main plug-in page with all the installed plug-in as shown in Figure 13a. As the user activates the plug-in (Figure 13b), a small pop-up is displayed on the bottom of screen, indicating the start of the NSD and CoAP server (Figure 13c).

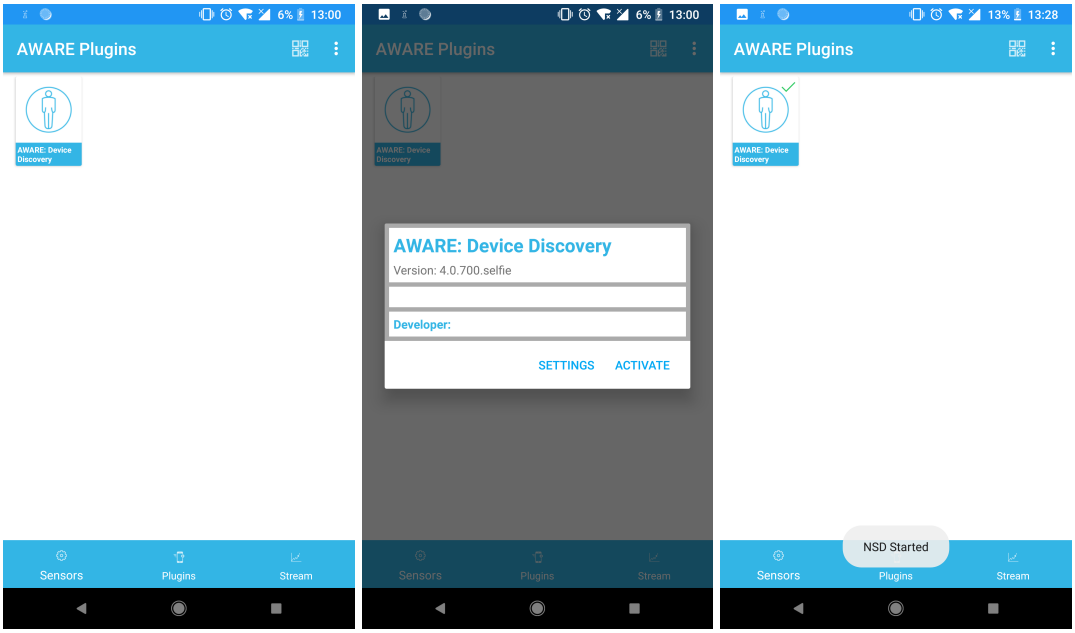

Figure 13. User Interface socialAWARE: plug-in. (From Left) (a) AWARE Plug-in page (b) Activation of Plugin (c) Pop-up

After activation the user switches to "Stream" page, that contains the list of the devices discovered using the NSD service (Figure 14a). The list is updated automatically whenever the plug-in finds another device. A new row is added with connection details of the discovered device (Figure 14b) and it is removed after that device is unregistered from the network.

After selecting a device the user is navigated to another page which shows the resources offered by the connected device. Once the user selects the required sensor, a new page loads and the sensor values received from the connected device are shown to the user in real time. In order to keep the plug-in simple, the sensor value text is changed automatically whenever the device receives the updated value.

Figure 14. User Interface socialAWARE: plug-in (From Left) (a) Main Plug-in page (b) List of devices discovered (c) List of available resources (d) & (e) Sensor Value page

4. EVALUATION

This chapter describes the experiments setup to evaluate the designed application. Seven experiments are designed to study and compare the performance and related issues of the protocols used. The purpose of the first two experiments is to determine the time required to discover and resolve the AWARE devices using NSD. The third experiment is conducted to monitor the performance of CoAP protocol under different environments (Bandwidth limitations). The next four experiments will compare CoAP with MQTT under different scenarios which are explained in the next section.

4.1. Setup

After a working version of the application had been implemented, testing was started to evaluate its performance. The evaluation of the plug-in was separated into different parts. The first part was designed to evaluate the performance of the service discovery protocol. The later part consists of multiple experiments to compare the performance of the CoAP protocol with the MQTT. Table 1 and Figure 15 shows the list and specification of the devices used in the experiment.

Table 1. Devices used in the Experiment

Device	Quantity	Specification	Operating System
Google Pixel XL	1	Quad-core Snapdragon 821 4GB RAM	Android 8.1
Motorola Moto G3	5	Quad-core 1.4 GHz, 2GB RAM	Android 6.0.1
Raspberry Pi 3	5	ARM Cortex-A53 1.2GHz, 1GB RAM	Raspbian Jessie
Asus RT-AC88U	1	802.11ac dual-band Wi-Fi router	NA

4.1.1. Service Discovery

The first study was designed to investigate the performance of the Network Service Discovery. Our goal is to study the time response of the NSD. We focus on searching the maximum response time to discover a given service and the maximum time taken to resolve the connection information of the discovered device. In this experiment, the first step was the registration of N number of mobile devices on the local network. After the registration, mobile device was used to discover the registered services. The last step was to resolve the connection information (IP and port) of the N devices. The time taken to discover and resolve the N number of devices was recorded and each experiment was performed five times and average was taken to reduce the effect of any error.

In the both experiments, Google Pixel XL was used to discover and resolve the connection information of the devices. In order to avoid any error, the NSD registration of the Google pixel XL and discovery for the Motorola Moto G3 was turned off. A custom script was written on Raspberry Pi 3 to register itself as a service on the network.

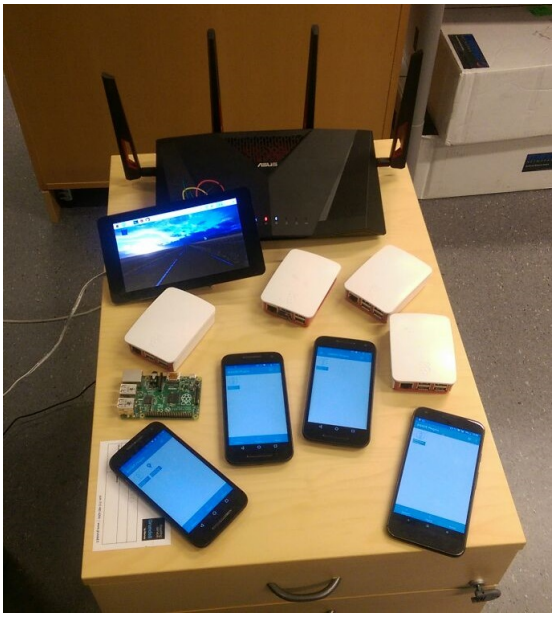

Figure 15. Devices used in the Experiment

In order to keep the services unique MAC address of the raspberry pi was concatenated with the AWARE String. The code for service registry is following:

```
<?xml version="1.0" standalone='no'?>
<!DOCTYPE service-group SYSTEM "avahi-service.dtd">
<service-group>
        <name replace-wildcards="yes">AWARE_%h</name>
        <service>
                <type>_http._tcp</type>
                <port>8567</port>
        </service>
</service-group>
```

4.1.2. Comparison between CoAP and MQTT

The second study was designed with multiple experiments to compare the performance of the CoAP and MQTT. The developed AWARE plug-in was used to calculate the bandwidth usage, latency and throughput of the CoAP in single and group communication mode. A custom application was also developed to record the same parameters for the MQTT protocol. In order to study the behavior of the protocols at different bandwidths, the experiment was performed in three different bandwidth scenarios, limiting the bandwidth at 128Kbps, 512Kbps and full connection. For further study, the effect of the message size on latency and throughput was also monitored. The payload of the sent packet was changed, ranging from 32 Bytes to 100 Kilo-Bytes. Google Pixel XL was used as the CoAP and MQTT server in their respective experiments. Motorola Moto G3 and Raspberry Pi 3 were used as the client. The ASUS RT-AC88U router was

used to limit the connection speed between the devices. The protocols were configured with settings defined in the Table 2.

Table 2. Protocol Settings for Evaluation

	CoAP	MQTT
Confirmation Method	CON Message	QoS 0
Block Size	1024 Bytes	NA
Maximum Timeout	1000 ms	1000ms

In the first experiment, latency of the CoAP protocol was calculated by recording the time taken by a packet for the round trip between the Server and the Client device. The time was started as the packet with a custom payload was sent from the Server and was stopped when the confirmation of the packet was received from the client device. The experiment was performed for three different speeds with seven different types of payloads. Each scenario was performed three times and average was taken. Latency of the MQTT was also calculated in the similar manner. The only difference between the experiments was the use of Raspberry Pi 3 as the Broker for MQTT.

The second experiment was to calculate the throughput of the both protocols. Data from the first experiments was used to calculate the throughput. The formula used to calculate the throughput is as following

$Throughput = totalNumberofbitstransfered/totaltimetaken$

The third experiment was performed to study the effect of group communication on the latency and throughput of the protocol. The experiment was setup using the same parameters from the first experiment except different number of clients were connected. In the first scenario, two clients were connected to the Server device and latency was measured. In second scenario, five different clients were connected and all were receiving the same packet from the server. The data recorded was then used to calculate the throughput.

In the last experiment the bandwidth used by each protocol was monitored. The server device was connected to a client and one thousand packet with a payload of 32 Bytes were sent between them. The server network usage was monitored using the Android DDMS client. The acknowledgement packets received by the server were also added in the calculation. The data collected was then used to calculate the average network usage for both protocols. This experiment was also performed three times and average value was taken.

5. RESULT

This chapter describes the results of the experiments performed in the previous section. It contains the detail of the data collected during the experiment and the different analysis done to understand the behavior and performance of the protocols.

5.1. Service Discovery

The first experiment was to determine the maximum time taken to discover a device on the network. After the data was collected from the ten devices, it was mapped to forecast the time it will take to discover up to thousand devices using linear regression. Figure 16 presents the visualization of the experiment.

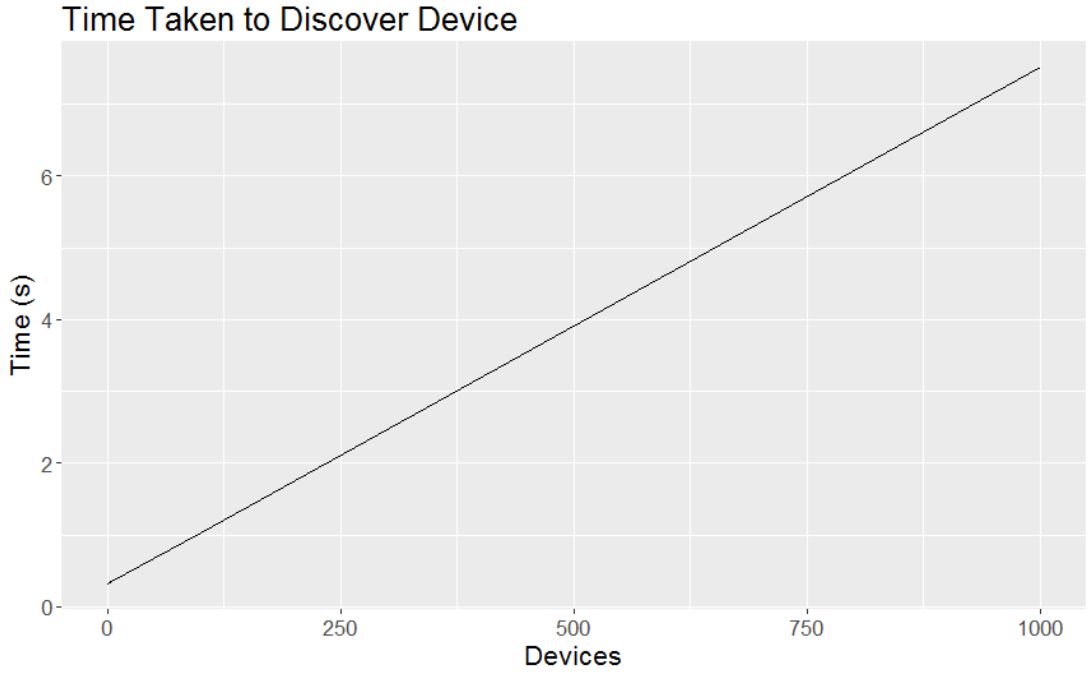

Figure 16. Time taken to discover AWARE devices

As seen from the graph, it takes about 300 milliseconds to discover the first device. The later devices are discovered faster than the first device, at an average time of 7 milliseconds per device. Using the linear regression, it was calculated that 1000 devices will require altogether about 7.5 seconds to get discovered by the mobile phone.

The second experiment was performed to determine the maximum time taken for a device to resolve the connection information of the discovered devices. The experiment was conducted for ten different devices and result was mapped for 1000 devices using linear regression. It can be deduced from the figure 17 that resolution of connection information requires more time than just discovering the service name. It takes about 390 seconds to resolve the information 1000 devices, with an average of 400 milliseconds for single device. As resolving of host information is a two-step process,

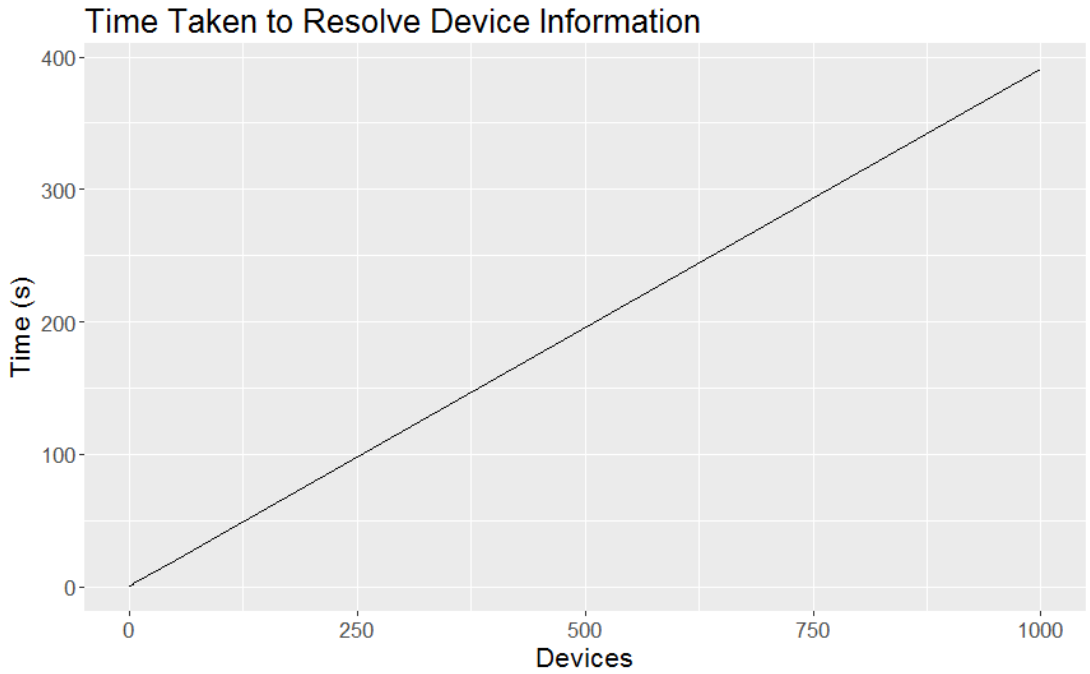

Figure 17. Time taken to Resolve AWARE devices information

i.e. searching for registered service on the network and performing a DNS lookup for connection information, it takes much more time than discovering of the service.

5.2. Comparison between CoAP and MQTT

After the performance evaluation of service discovery protocol, the next study was to compare the performance of the CoAP and MQTT protocols.

5.2.1. Latency

One-to-One Communication

After the collection of data, the results from the three scenarios were plotted. Figure 18 (a) and (b) shows the comparison between CoAP and MQTT at 128Kbps and 512Kbps.

Our result above shows the latency comparison at low bandwidth. It suggests that CoAP performs better with a smaller message size in low bandwidth environment compared to MQTT. CoAP packet with a payload of 32 Bytes takes around 1200 milliseconds to transfer from server to client at the speed of 128Kbps. It also including the time taken to receive the acknowledgement packet by the server device. This behavior of CoAP remains same till the packet size is 1 kilobyte after which the latency begins to increase rapidly. On the other hand, MQTT take around 1600 milliseconds for the same message size. Latency for the MQTT increases at a slower rate compared to CoAP, intersecting the line at around 7.5 Kilobytes, meaning the time taken for both

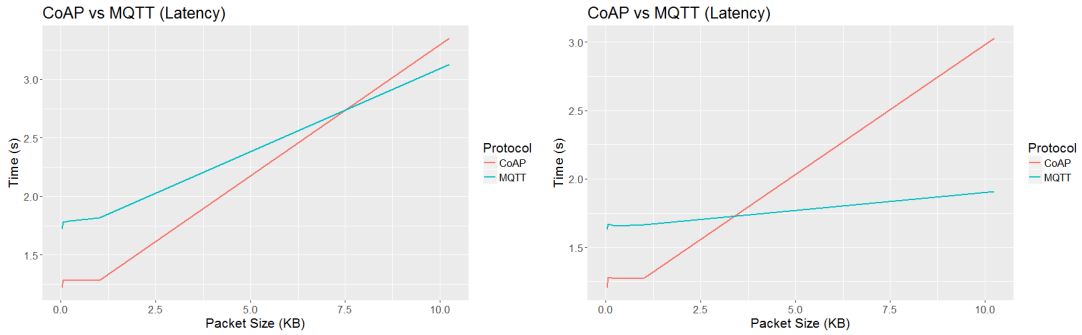

Figure 18. Latency comparison between CoAP and MQTT (From left) (a) 128Kbps (b) 512Kbps

protocols at this message size is same i.e. around 2700 milliseconds. After this the latency for CoAP is increase at a faster rate than MQTT, making MQTT better at bigger payload size.

A similar kind of trend can be seen in figure 18 (b), when the same experiment was repeated at 512Kbps. Latency for the both protocols decreased but MQTT performed better than 128Kbps. At this speed, the latency is same at around 3 Kilobytes of message size which is also lower than the previous experiment.

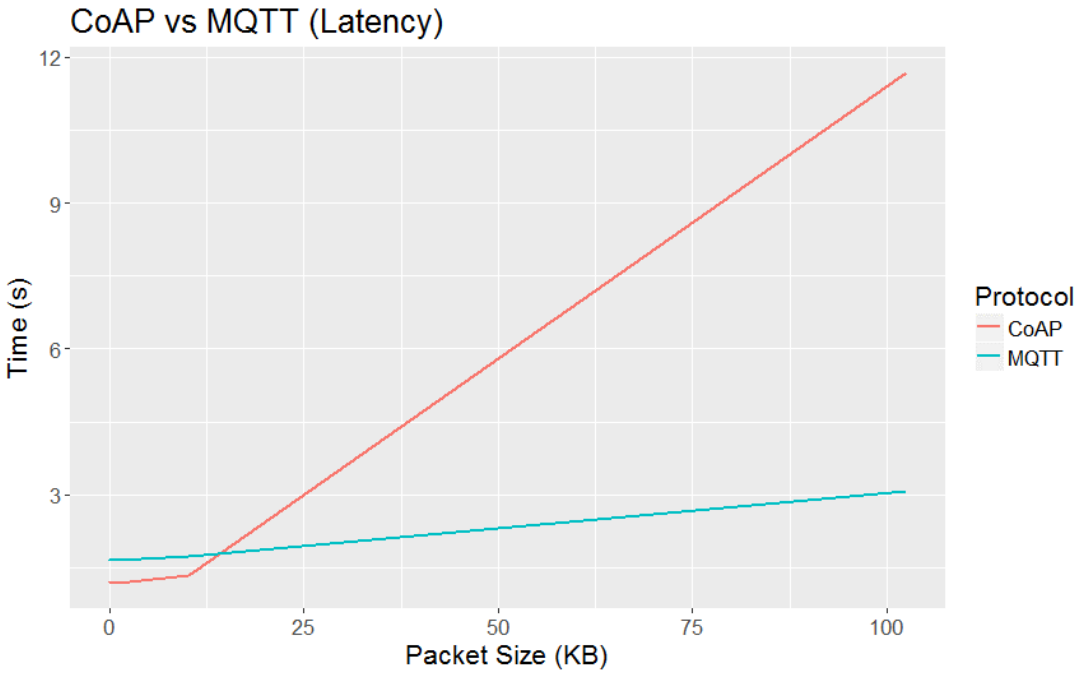

Figure 19. Latency comparison between CoAP and MQTT (Full Speed)

The last scenario in this experiment was to determine latency for both protocols without any bandwidth limitations. The above figure 19 shows the visual representation of results collected. At smaller message size CoAP is faster but as the packet size is increased the latency for CoAP increases and overall MQTT performs much faster.

It took around 11 seconds for a packet of 100 kilobytes, for a round trip when using COAP, while it took around 3 seconds for MQTT to deliver the same packet.

One-to-Many Communication

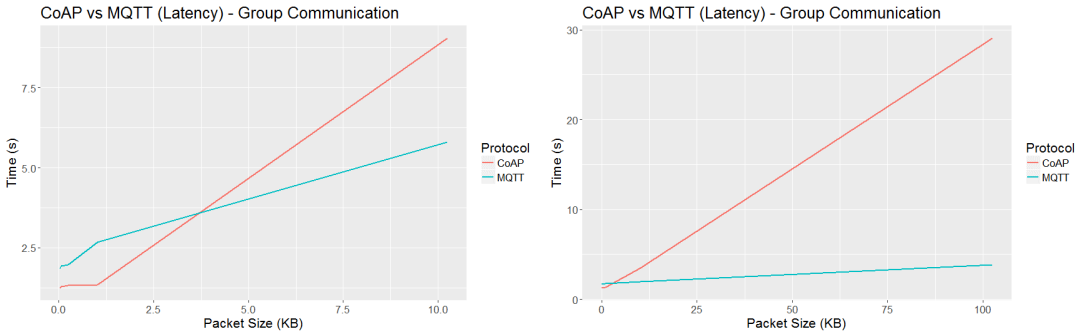

Figure 20. Latency comparison between CoAP and MQTT One-to-Many Communication (From Left) (a)128kbps (b) Full Speed

After the comparison of latency of CoAP and MQTT in one-to-one communication mode, the next experiment was to look at one-to-many communication. Figure 20 (a) shows the latency results of one-to-many communication at 128 Kbps and full bandwidth. Again at low bandwidth CoAP performed better when the message size is lower than 3.5 Kilobytes but after that MQTT takes less time to deliver the message. At faster speed (figure 20 (b)), MQTT was quicker by transferring the message of 100 Kilobytes in around 4 seconds and CoAP took around 29 seconds for similar task.

5.2.2. Throughput

One-to-One Communication

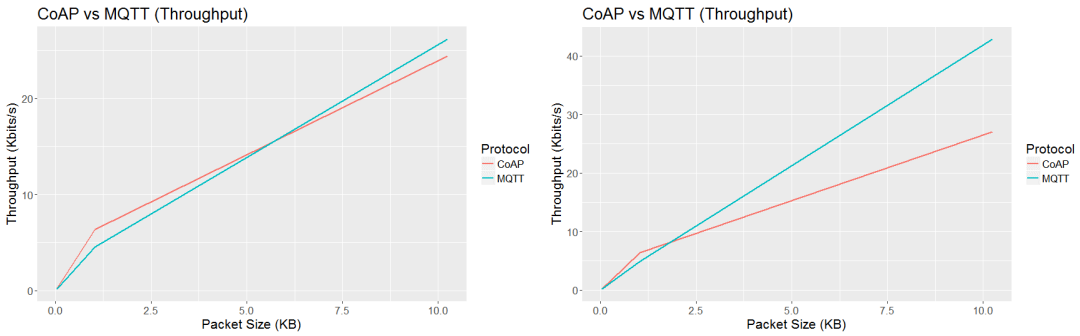

Figure 21. Throughput comparison between CoAP and MQTT (From Left) (a)128kbps (b) 512kbps

The second experiment was conducted to compare the throughput of the both protocols. Figure 21 shows the comparison of the throughput of the both protocols at

low bandwidth. At 128Kbps, both protocols showed almost same kind of result with CoAP having a higher throughput with smaller payload but MQTT performance improved with the message size. On the other side, the difference between the both protocols throughput is bigger at 512 Kbps, with MQTT achieving around 42 Kbits/s while CoAP reached 27 Kbits/s for the payload of 10 kilobytes.

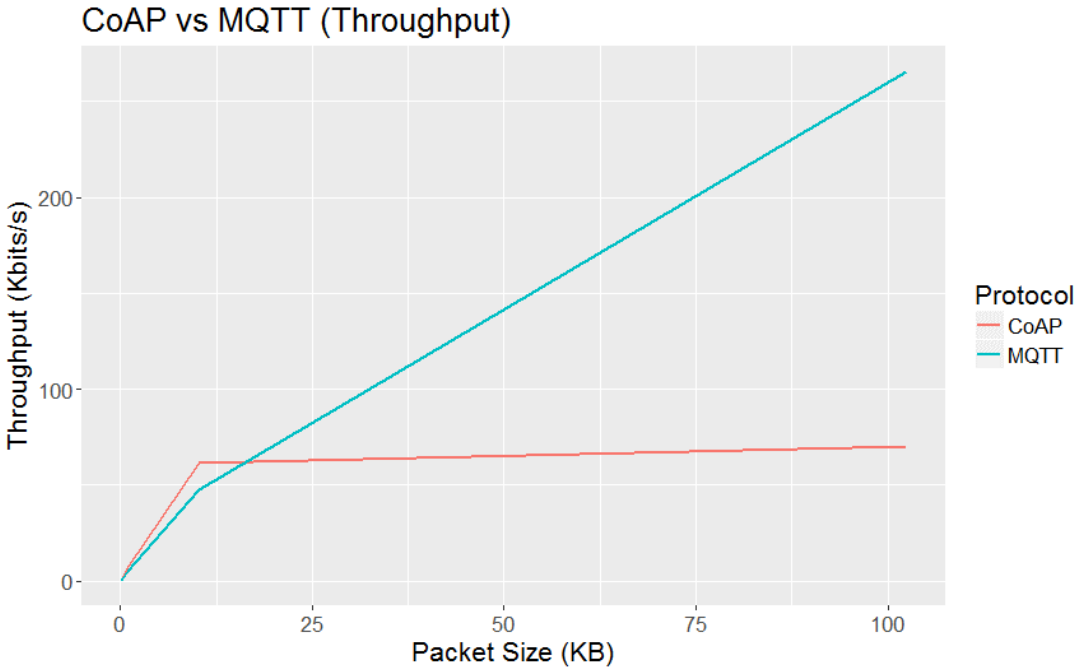

Figure 22. Throughput comparison between CoAP and MQTT (full speed)

The third scenario in this experiment was almost the replication of previous scenario but with no bandwidth limit. As seen from the figure 22, CoAP performed better at start by reaching the maximum throughput of 70 kbits/s but as the packet size was increased the throughput of the CoAP remained constant. MQTT had a lower throughput in the start but after increasing the packet size the throughput also increased. MQTT achieved a maximum throughput of 260 Kbits/s for 100 Kilobyte and it keeps on increasing with the packet size.

One-to-Many Communication

The third experiment in this study was done to compare the throughput of both protocols in one-to-many communication. Figure 23 shows the visual representation of the experiment results of the one-to-many communication. At low bandwidth, CoAP had a higher throughput till the message size reached 4 Kilobytes but after that MQTT performed better. In the figure 23 (b), MQTT was quicker from the start, it transferred the message of 100 Kilobytes at the rate of 220 Kbits/s while CoAP had a constant throughput of 25 Kbits/s.

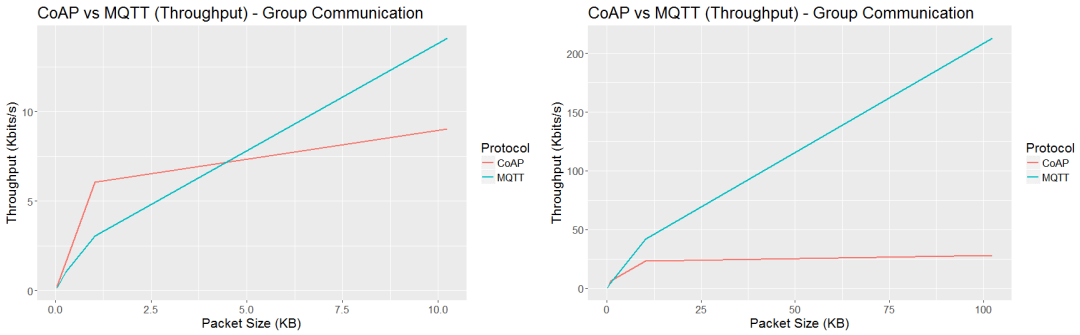

Figure 23. Throughput comparison between CoAP and MQTT, One-to-Many Communication (From Left) (a) 128kbps (b) Full Speed

5.2.3. Network Usage

The last experiment was to compare the network usage of the CoAP with MQTT. Table 3 shows the results from the experiment performed.

Table 3. Network usage comparison

Protocol	Total Packets	Total Bytes	Average size per packet
CoAP	1000	44720	44.72
MQTT	1000	62333	62.33

As seen from the table above, average size per packet of CoAP is lower than MQTT. Therefore, CoAP uses less bandwidth for the same 32 byte payload.

6. DISCUSSION AND LIMITATIONS

The purpose of this thesis was to develop an AWARE plug-in and highlight the advantages and disadvantages of the different protocols used. After the system design and implementation of the application, multiple experiments were conducted to analyze the performance of the protocols. This chapter discusses in detail the results of those evaluations and some points based on the analysis. In addition, the limitations and future work of this thesis is also presented.

6.1. Device Discovery

Our first study was to examine the performance of the device discovery protocol. In our study we found out that the time required to discover a device using NSD takes around 300 milliseconds. These finding verifies that NSD takes almost same time as other zero-configuration implementations [45]. This means that tens of hundreds of AWARE devices on any local network can be discovered with-in few seconds. The use of service discovery helps a user with little technical knowledge in setting up, configuring and maintaining a small wireless sensor network.

The use of NSD also arises some limitations for our application. First, NSD makes the device visible to all the other devices on the network, raising the issue of security. As the purpose of the application is to deploy a sensor network in closed environment, perhaps the easiest security model for this scenario is to have no security at all and simply rely on physical security mechanisms [74]. Second, NSD is running all the time in background to discover new devices and keep track of the discovered ones, it consumes a lot of battery. This limitation cannot be eliminated but its effects can be minimized. Battery efficiency can be improved by automatic turn-off of NSD when application is inactive for few minutes. It can also be improved by adding an option of manually setting up the interval time between device discovery scans.

The network service discovery can be further improved by the implementation of the service discovery capable DNS server over the internet. This will allow AWARE devices to discover and resolve connection information of other mobiles all over the world.

6.2. CoAP

The second study was conducted to examine the performance of CoAP and compare it with MQTT. In terms of transferring the data, our findings from the first experiments showed that CoAP is faster and better when the message size is small. These findings can be verified by previous research that reports CoAP takes less time than MQTT in start of transmission [31].Based on the results from the second experiment, it can also be concluded that one-to-many communication between the devices in CoAP is also slightly better than MQTT in this scenario. The finding from the experiments performed at low bandwidth confirms that CoAP has lower latency and higher throughput for smaller packets. The result from the network usage experiment proves that CoAP is more lightweight than MQTT making it more suitable for mobile devices.

The limitations of CoAP are also fully acknowledged. From above analysis, it can be deduced that CoAP is not suitable to transfer large messages or files. As the application is used to transfer the sensor values in real time and average size of a sensor value is less than 100 bytes, it make CoAP suitable for this application. Another problem with CoAP is the use of UDP as communication protocol, as it does not assure reliable communication between devices. To tackle this problem we use the conformable messages that retransmits the data if ACK packet is not received.

CoAP is mainly used for one-to-one communication but it also has support of observing the resources for group communication. Multiple connection to CoAP increases the latency and network usage, as connection between each device is independent. This also increases the CPU and memory usage of the application making it energy inefficient. This can be improved in future by manually setting the connections limit made to the host device.

In the last, our implementation of CoAP works only in local area network, needing a proxy server to connect it with other devices outside the network. A good improvement for our plug-in will be the implementation of universal proxy server for all the AWARE devices to communicate with other devices beyond the local network.

7. CONCLUSION

To conclude, the work presented in this thesis focuses on the implementation of an AWARE plug-in that helps the user to discover the devices using AWARE. It also focuses on using the CoAP protocol for efficient machine-to machine communication between the discovered devices. The plug-in was designed to run with AWARE application on Android mobiles. We have discussed the underlying architecture and the building blocks of the plug-in. In our implementation, we also discussed about the use of different parameters and the sequence flow of the plug-in. We kept the user interface of the plug-in as simple as possible so that all users can use it easily.

After the implementation, we conducted multiple experiments to evaluate the performance of the plug-in on different metrics. We later analyzed the data collected from the experiments. Our finding showed that service discovery require few hundred milliseconds to establish a connection between devices. The detailed performance analysis of the device discovery and resolution proved that the zero-configuration can be adapted with reasonable performance latencies on mobile devices, paving the way for the mobile sensor network.

As a transfer protocol, CoAP and MQTT both shows promising result in terms of performance. The experiments showed that CoAP was better suited for low latency networks. It performs better than MQTT when the message size was small making it suitable for the transfer of the sensor data. On the other hand, MQTT performs better at faster bandwidths and group communications. In addition, network usage experiment showed that CoAP was more lightweight than MQTT. Both protocols have been shown to be viable alternatives as transfer protocols for mobile sensor networks.

The developed AWARE plug-in provides great utility and can offer many research possibilities. It can help users to setup a wireless sensing network without much technical capabilities. The work also provides an insight of the IoT protocols and their comparison in different scenarios. We hope our findings would be of significant importance and will help in further improvements of the AWARE application.

8. REFERENCES

[1] (Accessed: 19.10.2017), What is aware. URL: http://www.awareframework.com/what-is-aware/.

[2] Tan L. & Wang N. (2010) Future internet: The internet of things. In: Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on, vol. 5, IEEE, vol. 5, pp. V5–376.

[3] IBM (2009), Smart china report.

[4] (Accessed: 17.05.2017), Web services architecture requirements. URL: https://www.w3.org/TR/wsa-reqs/.

[5] Zeng L., Benatallah B., Ngu A.H., Dumas M., Kalagnanam J. & Chang H. (2004) Qos-aware middleware for web services composition. IEEE Transactions on software engineering 30, pp. 311–327.

[6] Curbera F., Duftler M., Khalaf R., Nagy W., Mukhi N. & Weerawarana S. (2002) Unraveling the web services web: an introduction to soap, wsdl, and uddi. IEEE Internet computing 6, pp. 86–93.

[7] Fielding R.T. (2000) Fielding dissertation: Chapter 5: Representational state transfer (rest). http://www. ics. uci. edu/˜ fielding/pubs/dissertation/rest_arch_style. htm .

[8] (Accessed: 26.10.2017), Representational state transfer (rest). URL: https://www.service-architecture.com/articles/web-services/representational_state_transfer_rest.html.

[9] (Accessed: 10.10.2017), Internet of things global standards initiative. URL: http://www.itu.int/en/ITU-T/gsi/iot/Pages/default.aspx.

[10] Xia F., Yang L.T., Wang L. & Vinel A. (2012) Internet of things. International Journal of Communication Systems 25, p. 1101.

[11] Jabbar S., Ullah F., Khalid S., Khan M. & Han K. (2017) Semantic interoperability in heterogeneous iot infrastructure for healthcare. Wireless Communications and Mobile Computing 2017.

[12] Shelby Z., Hartke K. & Bormann C. (2014) The constrained application protocol (coap) .

[13] Colitti W., Steenhaut K., De Caro N., Buta B. & Dobrota V. (2011) Evaluation of constrained application protocol for wireless sensor networks. In: Local & Metropolitan Area Networks (LANMAN), 2011 18th IEEE Workshop on, IEEE, pp. 1–6.

[14] Bormann C. & Shelby Z. (2011) Blockwise transfers in coap. draft-ietf-core-block-04 (work in progress) .

[15] Shelby Z. (2012) Constrained restful environments (core) link format .

[16] Hartke K. & Shelby Z. (2011) Observing resources in coap. draft-ietf-core-observe-02 (work in progress) .

[17] Brachmann M., Garcia-Morchon O. & Kirsche M. (2011) Security for practical coap applications: Issues and solution approaches. GI/ITG KuVS Fachgesprch Sensornetze (FGSN). Universitt Stuttgart .

[18] (Accessed: 08.10.2017), Libcoap: C-implementation of coap. URL: https://libcoap.net/.

[19] (Accessed: 08.10.2017), Californium (cf) coap framework in java. URL: https://eclipse.org/californium/.

[20] Iglesias-Urkia M., Orive A. & Urbieta A. (2017) Analysis of coap implementations for industrial internet of things: A survey. Procedia Computer Science 109, pp. 188–195.

[21] (Accessed: 08.10.2017), Zero configuration networking (zeroconf). URL: http://www.zeroconf.org/.

[22] (Accessed: 08.10.2017), Bonjour for developers. URL: https://developer.apple.com/bonjour/.

[23] (Accessed: 26.10.2017), Using network service discovery. URL: https://developer.android.com/training/connect-devices-wirelessly/nsd.html.

[24] Cheshire S. & Krochmal M. (2013) Multicast dns. Tech. rep.

[25] Srinivasan S. & Schulzrinne H. (2007) Bonswing: A gui framework for ad-hoc applications using service discovery. In: Proceedings of the 2007 ACM CoNEXT conference, ACM, p. 36.

[26] (Accessed: 09.10.2017), Mqtt. URL: http://mqtt.org/.

[27] Al-Fuqaha A., Guizani M., Mohammadi M., Aledhari M. & Ayyash M. (2015) Internet of things: A survey on enabling technologies, protocols, and applications. IEEE Communications Surveys & Tutorials 17, pp. 2347–2376.

[28] Hunkeler U., Truong H.L. & Stanford-Clark A. (2008) Mqtt-s—a publish/subscribe protocol for wireless sensor networks. In: Communication systems software and middleware and workshops, 2008. comsware 2008. 3rd international conference on, IEEE, pp. 791–798.

[29] Ferreira D., Kostakos V. & Dey A.K. (2015) Aware: mobile context instrumentation framework. Frontiers in ICT 2, p. 6.

[30] Villaverde B.C., Pesch D., Alberola R.D.P., Fedor S. & Boubekeur M. (2012) Constrained application protocol for low power embedded networks: A survey. In: Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on, IEEE, pp. 702–707.

[31] Amaran M.H., Noh N.A.M., Rohmad M.S. & Hashim H. (2015) A comparison of lightweight communication protocols in robotic applications. Procedia Computer Science 76, pp. 400–405.

[32] Colitti W., Steenhaut K. & De Caro N. (2011) Integrating wireless sensor networks with the web. Extending the Internet to Low power and Lossy Networks (IP+ SN 2011) .

[33] Bormann C., Castellani A.P. & Shelby Z. (2012) Coap: An application protocol for billions of tiny internet nodes. IEEE Internet Computing 16, pp. 62–67.

[34] Kovatsch M., Duquennoy S. & Dunkels A. (2011) A low-power coap for contiki. In: Mobile Adhoc and Sensor Systems (MASS), 2011 IEEE 8th International Conference on, IEEE, pp. 855–860.

[35] Dunkels A., Gronvall B. & Voigt T. (2004) Contiki-a lightweight and flexible operating system for tiny networked sensors. In: Local Computer Networks, 2004. 29th Annual IEEE International Conference on, IEEE, pp. 455–462.

[36] Kovatsch M. (2011) Firm firmware and apps for the internet of things. In: Proceedings of the 2nd Workshop on Software Engineering for Sensor Network Applications, ACM, pp. 61–62.

[37] Lerche C., Hartke K. & Kovatsch M. (2012) Industry adoption of the internet of things: A constrained application protocol survey. In: Emerging Technologies & Factory Automation (ETFA), 2012 IEEE 17th Conference On, IEEE, pp. 1–6.

[38] Kovatsch M. (2011) Demo abstract: Human-coap interaction with copper. In: Distributed Computing in Sensor Systems and Workshops (DCOSS), 2011 International Conference on, IEEE, pp. 1–2.

[39] Schor L., Sommer P. & Wattenhofer R. (2009) Towards a zero-configuration wireless sensor network architecture for smart buildings. In: Proceedings of the First ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings, ACM, pp. 31–36.

[40] Rahman A., Dijk E. et al. (2012) Group communication for coap. draft-ietf-core-groupcomm-00 (work in progress) .

[41] Moritz G., Golatowski F. & Timmermann D. (2011) A lightweight soap over coap transport binding for resource constraint networks. In: Mobile Adhoc and Sensor Systems (MASS), 2011 IEEE 8th International Conference on, IEEE, pp. 861–866.

[42] (Accessed: 11.10.2017), Welcome to avahi. URL: `http://avahi.org/`.

[43] Gwizdz G. (2013) Zero configuration networking and communication using ios and android .

[44] Klauck R. & Kirsche M. (2012) Bonjour contiki: A case study of a dns-based discovery service for the internet of things. Ad-hoc, Mobile, and Wireless Networks , pp. 316–329.

[45] Siddiqui F., Zeadally S., Kacem T. & Fowler S. (2012) Zero configuration networking: Implementation, performance, and security. Computers & electrical engineering 38, pp. 1129–1145.

[46] Bardin J., Lalanda P. & Escoffier C. (2010) Towards an automatic integration of heterogeneous services and devices. In: Services Computing Conference (APSCC), 2010 IEEE Asia-Pacific, IEEE, pp. 171–178.

[47] Siljanovski A., Sehgal A. & Schonwalder J. (2014) Service discovery in resource constrained networks using multicast dns. In: Networks and Communications (EuCNC), 2014 European Conference on, IEEE, pp. 1–5.

[48] Schönwälder J., Tsou T. & Sarikaya B. (2011) Protocol profiles for constrained devices. In: Proceedings of the IAB Workshop on Interconnecting Smart Objects with the Internet (February 2011).

[49] (Accessed: 11.10.2017), Arduino ethernet bonjour. URL: http://gkaindl.com/software/arduino-ethernet/bonjour.

[50] (Accessed: 26.11.2017), Zero-configuration. URL: http://slideplayer.com/slide/7456746/.

[51] Dunkels A. et al. (2009) Efficient application integration in ip-based sensor networks. In: Proceedings of the First ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings, ACM, pp. 43–48.

[52] Östmark Å., Lindgren P., Halteren A.v. & Meppelink L. (2006) Service and device discovery of nodes in a wireless sensor network. In: IEEE Consumer Communications and Networking Conference: 08/01/2006-10/01/2006, IEEE Communications Society, pp. 218–222.

[53] Steinberg D.H. & Cheshire S. (2005) Zero Configuration Networking: The Definitive Guide: The Definitive Guide. " O'Reilly Media, Inc.".

[54] (Accessed: 26.11.2017), Cisco wide area bonjour application. URL: https://www.cisco.com/c/en/us/td/docs/cloud-systems-management/application-policy-infrastructure-controller-enterprise-module/1-5-x/bonjour/b_Cisco_Wide_Area_Bonjour_User_Guide/b_Cisco_SD_Bonjour_Solution_Guide_chapter_01.html.

[55] Lee J.W., Schulzrinne H., Kellerer W. & Despotovic Z. (2007) z2z: Discovering zeroconf services beyond local link. In: Globecom Workshops, 2007 IEEE, IEEE, pp. 1–7.

[56] Al-Karaki J.N. & Kamal A.E. (2004) Routing techniques in wireless sensor networks: a survey. IEEE wireless communications 11, pp. 6–28.

[57] Yick J., Mukherjee B. & Ghosal D. (2008) Wireless sensor network survey. Computer networks 52, pp. 2292–2330.

[58] Mainetti L., Patrono L. & Vilei A. (2011) Evolution of wireless sensor networks towards the internet of things: A survey. In: Software, Telecommunications and Computer Networks (SoftCOM), 2011 19th International Conference on, IEEE, pp. 1–6.

[59] Alliance Z. (2010), Zigbee smart energy 2.0 draft 0.7 public application profile.

[60] Kushalnagar N., Montenegro G. & Schumacher C. (2007) Ipv6 over low-power wireless personal area networks (6lowpans): overview, assumptions, problem statement, and goals. Tech. rep.

[61] Dawson-Haggerty S., Jiang X., Tolle G., Ortiz J. & Culler D. (2010) smap: a simple measurement and actuation profile for physical information. In: Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems, ACM, pp. 197–210.

[62] Kovatsch M., Weiss M. & Guinard D. (2010) Embedding internet technology for home automation. In: Emerging Technologies and Factory Automation (ETFA), 2010 IEEE Conference on, IEEE, pp. 1–8.

[63] Castellani A.P., Bui N., Casari P., Rossi M., Shelby Z. & Zorzi M. (2010) Architecture and protocols for the internet of things: A case study. In: Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010 8th IEEE International Conference on, IEEE, pp. 678–683.

[64] Aberer K., Hauswirth M. & Salehi A. (2006) A middleware for fast and flexible sensor network deployment. In: Proceedings of the 32nd international conference on Very large data bases, VLDB Endowment, pp. 1199–1202.

[65] Munir S.A., Ren B., Jiao W., Wang B., Xie D. & Ma J. (2007) Mobile wireless sensor network: Architecture and enabling technologies for ubiquitous computing. In: Advanced Information Networking and Applications Workshops, 2007, AINAW'07. 21st International Conference on, vol. 2, IEEE, vol. 2, pp. 113–120.

[66] Liu B., Brass P., Dousse O., Nain P. & Towsley D. (2005) Mobility improves coverage of sensor networks. In: Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing, ACM, pp. 300–308.

[67] Chen C. & Ma J. (2006) Memosen: multi-radio enabled mobile wireless sensor network. In: Advanced Information Networking and Applications, 2006. AINA 2006. 20th International Conference on, vol. 2, IEEE, vol. 2, pp. 5–pp.

[68] Rezazadeh J. (2012) Mobile wireles sensor networks overview. International Journal of Computer Communications and Networks (IJCCN) 2.

[69] Li D., Wong K.D., Hu Y.H. & Sayeed A.M. (2002) Detection, classification, and tracking of targets. IEEE signal processing magazine 19, pp. 17–29.

[70] Sinopoli B., Sharp C., Schenato L., Schaffert S. & Sastry S.S. (2003) Distributed control applications within sensor networks. Proceedings of the IEEE 91, pp. 1235–1246.

[71] Rytter A. (1993) Vibrational based inspection of civil engineering structures. Ph.D. thesis, Dept. of Building Technology and Structural Engineering, Aalborg University.

[72] Kintner-Meyer M. & Brambley M.R. (2002) Pros & cons of wireless. ASHRAE journal 44, p. 54.

[73] (Accessed: 17.11.2017), Class diagram. URL: https://en.wikipedia.org/wiki/Class_diagram.

[74] Toivanen H. (2001), Secure zero configuration.

9. APPENDICES

Appendix 1. Class Diagram of socialAWARE

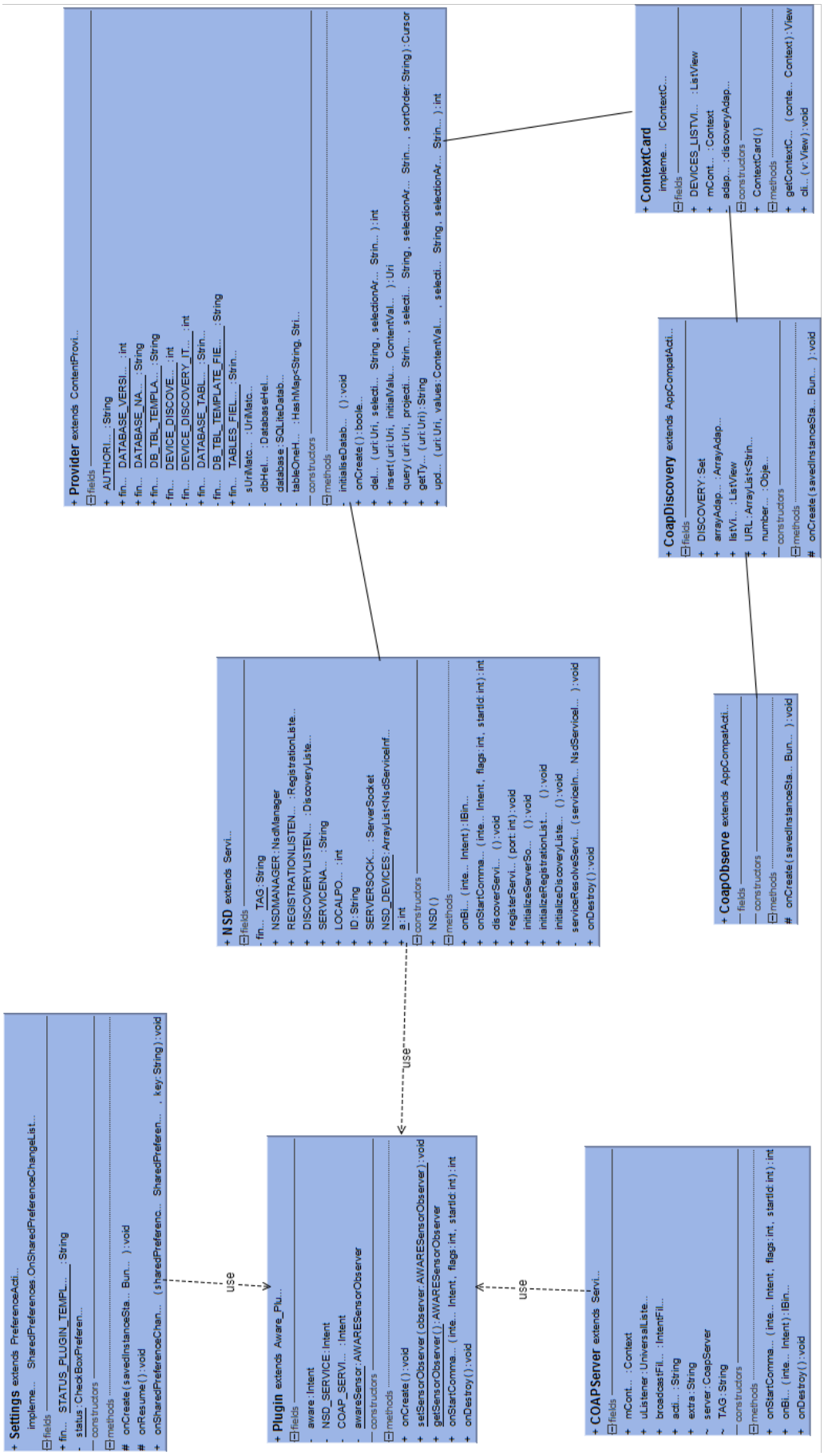

Figure 24. Class Diagram for socialAWARE plugin