International Journal of Ubiquitous Computing
Vol. 4 No. 1 (January-June, 2018)

LIGHTWEIGHT MOBILE WEB SERVICE PROVISIONING FOR THE INTERNET OF THINGS MEDIATION

Mohan Liyanage, Chii Chang and Satish Narayana Srirama

Mobile & Cloud Lab, Institute of Computer Science, University of Tartu, Ülikooli 17 - 324, Tartu 50090, Estonia

Received: 23rd March 2017 Revised: 14th April 2017 Accepted: 10th September 2017

ABSTRACT

Emerging sensor-embedded smartphones motivated the mobile Internet of Things research. With the integrated embedded hardware and software sensor components, and mobile network technologies, smartphones are capable of providing various environmental context information via embedded mobile device-hosted Web services (MWS). MWS enhances the capability of various mobile sensing applications such as mobile crowdsensing, real time mobile health monitoring, mobile social network in proximity and so on. Although recent smartphones are quite capable in terms of mobile data transmission speed and computation power, the frequent usage of 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. Although numerous previous researchers have tried to overcome the resource intensive issues in mobile embedded service provisioning domain, most of the efforts were constrained because of the underlying resource intensive technologies. This paper presents a lightweight mobile Web service provisioning framework for mobile sensing which utilises the protocols that were designed for constrained Internet of Things environment. The prototype experimental results show that the proposed framework can provide higher throughput and less resource consumption than the traditional mobile Web service frameworks.

KEYWORDS

Mobile Web service; CoAP; Llightweight; Cconstrained Service Provider

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 (MWS), where the mobile terminals are being used as both Web service clients and providers (which is termed-Mobile Hosts [1]). Mobile Web

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

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

service provisioning has been utilized in numerous fields such as Location Based Services (LBS) [2], Mobile Health Services [3], Mobile Social Networking applications [4], etc.
Although recent smartphones has enough processing power and capable of high speed data transmission, the challenge of resource constraints still exists when providing MWS. The battery power of the mobile device will drain quickly due to the extensive usage of the high performance multi-core mobile CPU and the high speed 3G/4G mobile Internet data transmission. Therefore, in past years, several lightweight mobile Web service provisioning approaches [5], [6] 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 Representational State Transfer (REST) based service provisioning instead of Simple Object Access Protocol (SOAP);
(2) Utilising external resources to enhance the overall performance. e.g., offloading the complex computational tasks to static Cloud [7] or to the Mobile Ad Hoc Cloud [8] and also using optimistic collaboration and scheduling to reduce the consumed bandwidth in order to serve energy [9], [10].

However, when considering MWS based sensing provisioning systems, they still have some other challenges that need to be addressed in order to increase their 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-Host 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 Host measure or prioritize the availability of the services?

To address these issues, in this paper, we extend ourprevious work [11] towards presenting a lightweight mobile Web service provisioning framework based on integrating a number of lightweight protocols including Constrained Application Protocol (CoAP) [12], Bluetooth Low Energy (BTLE) [13] and Efficient XML Interchange (EXI) [14]. 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.

The experimental results, which focused on the performance comparison between the traditional MWS and the proposed lightweight MWS, have shown the efficiency of the proposed lightweight MWS framework in terms of resource consumption and service provisioning throughput.

The rest of the paper is organized as follows: Section 2 provides background on applications of Mobile Host and comparison of related work. Section 3 discusses protocols related to service provisioning. Section 4 presents the design details of the proposed framework followed by the description of the main components of the MWS. Section 5 provides the implementation details of the prototype and Section 6 presents the performance evaluation of the proposed protocol stack. The paper concludes in Section 7 along with future research directions.

2. BACKGROUND AND RELATED WORK

This section explores the applications of the mobile host and state-of-art research on the mobile Web service provisioning.

2.1 APPLICATIONS OF MOBILE HOST

MWS has been utilized in different application domains over the past years.

Berger et al. [15] were successful in provisioning a SOAP-based retail Web application where a mobile device presents wallet services to an electronic check-out at kiosk. The kiosk station uses mobile wallet service hosted by the customer mobile devices that delivers a complete payment transaction within the mobile device, the retail store system and the bank system. Two other fascinating applications are a) Mobile Picture album with the location information service and b) Cooperative Journalism, where mobile host engages the coordination of journalists and their associations, which are presented in the work by Srirama et al. [16], and in [17] respectively. MobiCrop [18] is an application which grants crop farmers to use their mobile devices to get information about utilizing pesticides. Another interesting mobile health application provided in [19] called SOPHRA, which supports healthcare professionals to access patients' medical information that is hosted on their mobile devices.

2.2 COMPARISON OF EXISTING MOBILE HOST FRAMEWORKS

Most of earlier approaches were based on providing a SOAP-based MWS architectures. The work in Srirama et al. [16], [1], [20] presented a lightweight mobile host with Personal Java implementation. In that work, authors utilised HTTP and TCP for the application and transport layer protocols respectively, and SOAP messages format with the WSDL as service description.

Gehlen and Pham [21] presented a P2P Web services on Java enable mobiles. In this architecture, each mobile node work as both client/server role which provides service in the distributed environment. The authors developed a mobile SOAP server that capable of providing XML Web services. In another work Srirama [22] , Srirama et al. [23] 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 applied to set up a virtual P2P network. However, JXTA uses either TCP or HTTP protocols to traverse network barriers, and also the work used WSDL for service discovery and SOAP message format. Moreover, Ou et al. [24] designed a Layered P2P architecture to provide Web services in converged cellular and ad-hoc network environments with the vertical tunneling model to speed up the service discovery. The framework consists of eight modules including a WSDL module for service publishing and a SOAP Module for generating and parsing SOAP messages between the client and the service provider.

In order to reduce the workload at the mobile host, Kim and Lee [25] proposed to 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, initiate by the service provider, when a request received and there is a change in its context information. For instance, changes of context information such as shortage of battery level, changes of the device location, etc., which stops it providing the service on its own. Even though, migration of the services reduces the workload at the mobile, still the communication based on the same traditional protocols like HTTP, TCP, WSDL and SOAP, which is heavyweight.

The RESTful Web service approach has been used in later mobile host frameworks, which is more appropriate for the constrained conditions because of their lightweight features. For

instance, Chang [26] proposed the context aware cache pre-fetching strategy for MP2P Web Service provisioning. Instead of XML formatted documents like SOAP, this frame work used RESTful architecture which is more suitable in the infrastructure-less decentralized MP2P environment. Also, Srirama and Paniagua [27] proposed a mobile host for Android devices based on OSGi (Open Services Gateway initiative) framework which is capable of providing services in a RESTful fashion using HTTP and XMPP.

Mohamed and Wijesekera [5] proposes a lightweight framework for providing Web services on mobile devices which based on the REST architecture with JSON data representation. Moreover, the framework consists of main elements such as "HTTP Request/Response Listener", "Deployment Agent","Request Processor" to integrate the Web ser- vices, and the "Security Agent" to provide security services for the Web services. In addition to that, another framework that enables context-aware services, utilizing a hybrid model of both Mobile P2P and JmDNS for service discovery presented by Charl van der [28]. Accordingly, the JSON formatted messages reduce the communication over- head between devices and the multicast interchange messages increasing the reliability of the proposed framework.

Although, 6LoWPAN protocols (i.e., IPv6 over Low Power Wireless Personal Area Networks) provide WSNs to use IP to communicate with other networks, the application protocols need to develop or modify in order to enable the fully device communication across the WSN settings. With this intention, use of the Devices Profile for Web Services (DPWS) as an application layer protocol in WSNs was introduced Moritz et al. [29]. The authors presented a SOAP-binding on top of the IETF Constrained Application Protocol (CoAP) coupled with Efficient XML Interchange (EXI) format to reduce overhead as well as to increase the efficient delivery of SOAP messages. However, the main shortcoming of using SOAP is that it needs heavy-weight parsers, which degrades the overall performance of the mobile Web server.

Doukas et al [30] presented a generic Mobile SDK that allows developers to easily integrate IoT protocols (i.e., WebSockets and MQTT) into their applications in order to communicate with cloud based IoT environment. The authors described how a mobile device can communicate to an IoT application with the generic Mobile Software Development Kit (SDK) called COMPOSE. The SDK enables developers to implement communication with devices and IoT services leveraging various IoT protocols, like MQTT, RESTful communication and WebSockets, etc.

Although, in the above mentioned frameworks, the service architecture has been switched from the SOAP to REST, but 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, should utilize lightweight application protocol such as CoAP, EXI, etc. which do not included in the existing MWS frameworks.

3. RELATED SERVICE PROVISIONING PROTOCOLS

This section describes the related protocols/technologies used in our framework. It consists of four major parts of MWS*: message format, data transmission, service description and service discovery mechanism.*

3.1 MESSAGE FORMAT

In MWS provisioning, reducing the payload size in data transmission phase is improve the performance and energy efficiency. Moreover, binary XML and JSON are common options in past works [31]. 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 binary format that improves encoding/decoding performance and significantly reduces bandwidth requirements [32]. Overall EXI has a high compression efficiency of 50 times when compared to other compression schemes [33].

3.2 DATA TRANSMISSION

Traditional Web service provisioning relies on SOAP over HTTP. Past MWS frameworks [26], [5], [27] tended to utilize Representational State Transfer (REST) [34] based service provisioning in order to reduce the resource consumption of mobile host.

3.3 REPRESENTATIONAL STATE TRANSFER

REST is a communication protocol which handles the client-server communication over HTTP. A Web service designed based on REST is termed---RESTful service. URIs (Uniform Resource Identifiers) are used to access the resources over the Web server. The part of REST URI contains the ‘‘resource’’ which is also called Uniform Resource Name (URN) that is requested by clients. RESTful applications use the basic HTTP methods (i.e. GET, POST, PUT and DELETE) to access the Web servers. Based on the experimental results [35],[27],[28] REST is more suitable for resource constrained mobile computing environments.

3.4 CONSTRAINED APPLICATION PROTOCOL (COAP)

CoAP is a RESTful Web transfer protocol that is designed for constrained environments [12]. CoAP provides a more compact message format mechanism, which introduces low overhead binary encoded header and reduces the message parsing complexity.CoAP is built on top of the User Datagram Protocol (UDP) that has significantly lower overhead with the multicast support contrast to the TCP.

3.5 LOCAL SERVICE DISCOVERY

Local service discovery in peer-to-peer (P2P) manner is one of the major features of MWS provisioning in many scenarios [23],[7]. Different to the global service discovery, which can be achieved by utilizing 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 mobile host might utilize Wi-Fi or Bluetooth to support the P2P-based service discovery mechanism. We proposed to use Bluetooth Low Energy (BTLE) protocol that was designed for applications which require low power consumption and low radio duty cycle.

3.6 SERVICE DESCRIPTION

Service description metadata plays an important role in dynamic service composition systems. By providing standard service description metadata (e.g. WSDL), software clients can automatically identify the operations provided by the service. In the following sections, we discuss three main service description approaches that are suitable for MWS provisioning.

3.7 WEB SERVICE DESCRIPTION LANGUAGE

Original WSDL was designed to describe SOAP-based Web services. However, WSDL 2.0 is capable of describing RESTful services. Following is a portion of a WSDL 2.0 document that was used to describe a RESTful operation provided by the mobile host. An alternative Web service

description beside 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[4].

3.8 SENSOR MODEL LANGUAGE (SENSORML)

SensorML[5] is a metadata-based resource description language. It provides XML encodings for describing sensors, actuators, and processes. The schema and namespace information are defined under the header of the SensorML document. Generally, the elements defined in the document are mainly depending on the services and resources associated with the devices. Additionally, SensorML supports semantic description, which is not provided by WSDL 2.0 by default.

4. SYSTEM DESIGN

4.1 OVERVIEW OF ARCHITECTURE

Error! Reference source not found. illustrates the architecture of the proposed MWS provisioning. In this environment, mobile host represents the Web service provider that is capable of providing various data to its clients. For example, the mobile host 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 communication protocol of the mobile host is mainly based on CoAP, and the message payload is in EXI format.

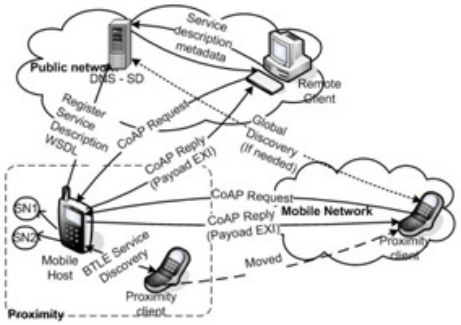

Figure 1. Mobile Web Service Provisioning Framework

The mobile host supports two types of service discovery: *global service discovery* and *local service discovery*. The global service discovery is realized by publishing its service description metadata (e.g., WSDL, WADL or SensorML) to a global service registry (e.g., DNS-SD). A remote client that knows the location of the service registry can discover the mobile host by requesting the service registry. Afterwards, the remote client can perform the regular service request to the mobile host by sending the CoAP service requests. The local service discovery is based on BTLE. As the **Error! Reference source not found.** shows, a client can discover the

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

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

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

mobile host in proximal range via BTLE. The mobile host will provide certain information via BTLE to help the client to identify the communication requirement. If the client moves out of the BTLE range with the mobile host, it can still communicate with the mobile host via a mobile Internet connection. The mobile host has been assigned a public static IP address from the mobile Internet service provider (e.g. Estonian TELE2 and EMT both provide this service). In the situations where the public IP address allocation is not possible, the mobile host can utilize proxy services [6], or hole-punching/relaying mechanisms [38].

4.2 SENSING SERVICE PROVISIONING

The mobile host can provide services based on time or resource allowance. For example, the user of the mobile host 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. Mobile host supports three types of sensing services:

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 mobile host will perform 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 mobile host will perform the sensing activity once, and provide the data to a specific endpoint based on the address described in the request message. Basic protocol stack.

4.3 BASIC PROTOCOL STACK

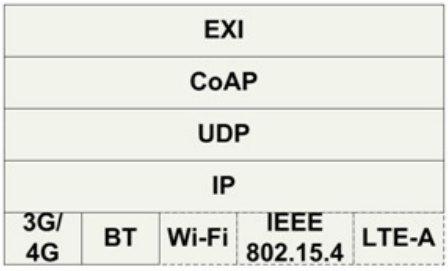

Figure 2. Protocol Stack of the Mobile Web Server

Error! Reference source not found. illustrates the conceptual design of the protocol stack used in the proposed mobile host. The proposed framework is mainly based on lightweight protocols to make the process simple. In the lower layer of the protocol stack, the mobile host can utilize numerous common protocols such as the 3G/4G mobile Internet (for global service interaction), Wi-Fi/Wi-Fi Direct, Bluetooth (BT)/BTLE, 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 utilizes CoAP that provides a solution with a compact header size, simple, lightweight, RESTful message exchange in between mobile host and clients over constrained network.

As mentioned in the earlier, the proposed mobile host also utilizes EXI to compress the size of the message payload. Since the design of the mobile host 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.

4.4 COMPONENTS OF THE MOBILE HOST

Error! Reference source not found. illustrates the component architecture of the proposed mobile host. It consists of following main components.

4.4.1 REQUEST/RESPONSE LISTENER

It is the first contact point for the clients to the mobile host. Once the mobile host 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.

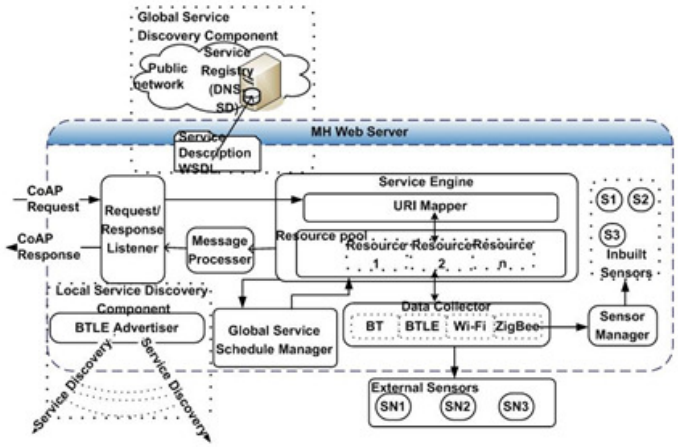

Figure 3. Main Components of Mobile Host Web Server

4.4.2 SERVICE ENGINE

This is the core module of the mobile host that coordinates all the parts of the mobile host. 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 analyzes 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.

4.4.3 DATA COLLECTOR

Data collector is responsible for collecting data from available resource providers. Such providers can be the inbuilt sensor components of the mobile host, 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.

4.4.4 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 mobile host framework utilizes 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.

4.4.5 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 minimize the energy consumption of the mobile host.

4.4.6 LOCAL SERVICE DISCOVERY COMPONENT

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

4.4.7 GLOBAL SERVICE DISCOVERY COMPONENT

To enable the mobile Web services provisioning across other networks, the mobile host 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.

4.4.8 GLOBAL SERVICE SCHEDULE MANAGER

Mobile host 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 1 (Global Sensing Service Schedule - GS)

GS is defined as a tuple (T , λ) where:
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, $|T| = 60$.
$\lambda: T \rightarrow S$ maps timestamps with the scheduled sensing service executions.
Mobile host 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.

4.5 SENSING SERVICE SCHEDULER MODEL

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

Definition 2 (Sensing Services Pool- SSP)

SSP describes the information of sensing services provided by the mobile host. It is defined as a tuple (S,ς,κ,ε)where:

S: is a set of sensing services
ς: $S \rightarrow R$ maps sensing services to sensing components (e.g.,GPS sensor, accelerometer camera, audio recorder, network signal browser etc.)
κ: $S \rightarrow S$ maps sensing services to conflict services
ε: $S \rightarrow E$ maps sensing services to the system resource usage sets (e.g. CPU, RAM, network transmission bandwidth usage, etc.)

Example 1 (Conflict Service)

Let s1 and s2 be two services. Let T` ⊆ T. Assume s1 has been requested by a real-time sensing request, which has its timestamp period withinT`, and s2 is not requested by any client. Suppose $s2 \in \kappa(s1)$, then s2 will be marked as unavailable duringT`.

4.5.1 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 (Request Processing - RP)

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

- N is a set of nodes
- $F \subseteq N \times N$is a set of flow relation rule
- $\tau : N \rightarrow Y$ 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*
- $P = \{p_1, \ldots p_n\}, n \in N$ 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 2 (Work Schedule Plan Handling)

Suppose a *RP*contains $P = \{p_1, \ldots p_n\}$. 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 *P*. It is based on the following rules:

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

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

4.5.2 ONE-TIME REQUEST PROCESSING

A one-time request involves 1...N sensing service invocation (services of *sType* nodes), which is denoted by S^{XR}, $S^{XR} = \{S_I^{XR} | \leq i \leq N\}$.

Let $t_x = \in T, and\ S_x = \lambda(t_x)$ a set of executed services at $t_x\ and\ S_x \cong now$, in which *now* denotes the current system time.

Let $S` = S^{XR} \cap S_x, if\ |S`| > 0\ then \forall s \in S`$, sensing service - *s* will be assigned for executing the activity. Let $S" = S^{XR} \setminus S_x$. For each $s \in S''$, 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.

4.5.3 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.

4.5.4 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 S^{tx} be a set of scheduled service executions at the timestamp $t_x \in T$, $S^{tx} = \{s_m^{tx} | 1 \leq m \leq N\}$. $k(s_m^{tx})$ denotes a set of conflict services of s_m^{tx}. Hence, a set of conflict services at timestamp - t_x (denoted by K^{tx}) will be: $K^{tx} = \cup_{m \in |S^{tx}|} k(s_m^{tx})$, and the available services at timestamp t_x (denoted by S^{tx}) will be $S^{tx} = S \setminus K^{tx}$.

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

Let $E^{sys} = \{e_0^{sys} | \leq 0 \leq N\}$, 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 S^{tx} be a set of sensing services assigned at $t_x = \in T$. For each service - $s_z \in S^{tx}$, its system resource consumption is found in $\varepsilon(s_z)$. Let $E^{s_z} = \varepsilon(s_z)$ in which $E^{s_z} = \{e_0^{s_z} | 1 \leq 0 \leq N\}$ in which the system resource denoted by $e_0^{s_z}$ and e_0^{sys} are the same, and let $ve_0^{s_z}$ be the usage

value of $e_0^{s_z}$ and ve_0^{sys} be the remaining value of e_0^{sys}. For each $t_x$the ve_0^{sys}after assigning S^{t_x}(denoted by $ve_0^{t_x}$) will be: $ve_0^{t_x} = ve_0^{sys} - \sum_{sz \in S^{t_x}} ve_0^{s_z}$.

Let $E^{t_x} = \{ v\, e_0^{t_x}$. Referring to the previous result, S^{tx} is a set of services that has been identified as available at t_x. For a service $s_y \in S^{t_x}$, let E^{s_y} be the system resource usage required by the service. If $\exists e_0^{s_y} \in E^{s_y}$, such that $ve_0^{s_y} > ve_0^{t_x}$, which indicates that the service s_y requires a higher usage value than the actual available resource value. Hence, the s_y is considered as unavailable at the time stamp t_x.

5. IMPLEMENTATION

This section describes the prototype implementation of the proposed framework.

5.1.1 MOBILE HOST

The mobile host was implemented on Google/LG Nexus 5 running Android version 5.0.1. The implementation is basically adapted from the JCoAP[6] that provides a Java API for the CoAP. The mobile host has also been tested on Raspberry Pi B+ and Nexus 9 tablets.

For the external sensory data collection we implemented Arduino based temperature sensor module which senses the ambient temperature and sends the data to the mobile host over the BTLE 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). We 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 mobile host needs to instantiate a new CoAP local endpoint. In our implementation, we defined four types of resources:

TmpResource - provides the current room temperature
LocationResource - provides current GPS details of the mobile host
AltitudeResource - provides current altitude information of the mobile host
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 mobile host.

5.1.2 EXI DATA PROCESS

We used ExiProcessor[7] the open source Java-based library that encodes text-XML files into binary EXI and decodes EXI files back to XML. 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, we managed to implement ExiProcessor on the Raspberry Pi with the 3G dongle for the mobile Internet connectivity.

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

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

5.1.3 LOCAL AND GLOBAL SERVICE DISCOVERY

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

6. PERFORMANCE EVALUATION

We designed a case study to investigate the performance of the proposed Web service framework.

6.1.1 EXPERIMENT 1

The mobile host implemented in Google/LG Nexus 5 running Android version 5.0.1 with the TELE2 4G mobile Internet connection. The clients simulate on a computer that connects to the University's Wi-Fi network. Clients request the altitude of the mobile host that replies the CoAP response message with the payload of 23 Bytes. We simulated different number of simultaneous client requests from 10 to 300 per second.

Then we measured the average throughput using the request/response ratio in each occasion. We repeated the same test case with the mobile host that use the conventional Web service framework (HTTP/REST) to contrast the lightweight in our mobile Web service framework. We also performed the same test case in the University Wi-Fi network to evaluate the performance of the framework further.

6.6.2 THROUGHPUT COMPARISON

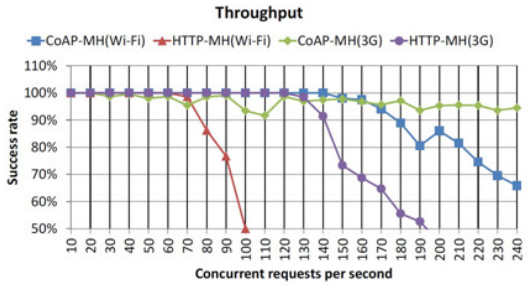

Figure 4. Throughput Of MWS

Error! Reference source not found.4. 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 requests per second and at least maintains 95% success rate up to 160 coinciding requests per second in the Wi-Fi network. Furthermore, we 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 can 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.

6.6.3 CPU USAGE COMPARISON

In order to record the CPU usage while the mobile host operating MWS, we utilised the Android Device Monitor that displays the CPU Load for the top applications running on the mobile device. As the **Error! Reference source not found.**. shows, the average CPU load is below

5% in the CoAP-mobile host. Conversely, in HTTP-mobile host, 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 mobile host application crashed because it is CPU intensive. On the other hand, CoAP-based mobile host can accommodate even a higher load from clients. Another interesting factor we observed is that the kernel CPU loads of both applications. In the HTTP-based mobile host, kernel CPU load is at a very high level while comparing with the CoAP-mobile host, which is as little as about 1%.

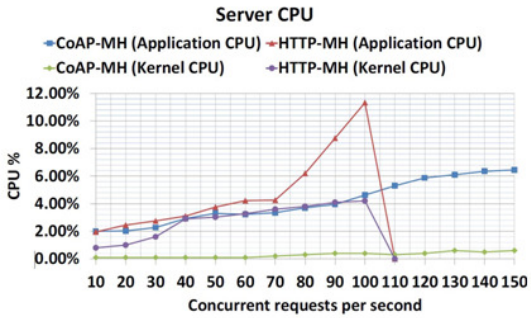

Figure 5. CPU load of MWS

6.6.4 ENERGY CONSUMPTION COMPARISON

To measure the energy consumption, we monitored total energy consumption of the device when the mobile host is running under the different loads. During the test the mobile host serves 20, 40, 60 and 80 clients per second with the size of payload from 100 bytes to 1000bytes. 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 the **Error! Reference source not found.**, 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 the both protocols as expected (**Error! Reference source not found.**. Chart (a)). However, we noticed that decrement of power consumption of the HTTP server at higher payloads (800 bytes & upwards). For instance, as shown in the **Error! Reference source not found.**6. 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 CoAP server, but slightly decreases in HTTP server. Furthermore, we can see that HTTP consumes less energy than CoAP when the payload is 1000 bytes. The reason that we observed is many sessions have been dropped at this point due to the limitations of the HTTP server which caused the drop of energy consumption.

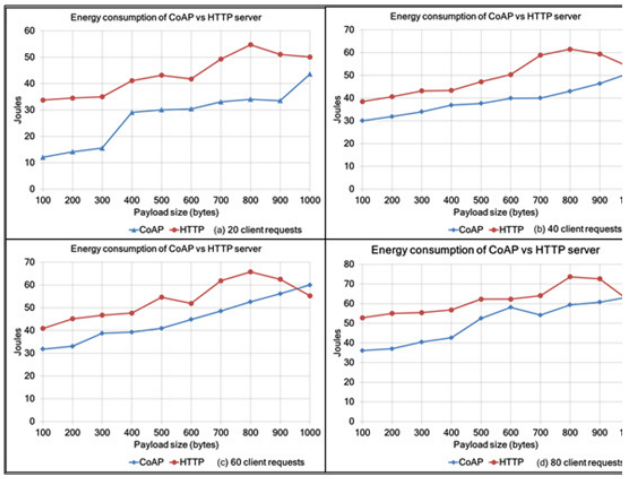

Figure 6. Comparison of Energy Consumption (Coap Vs HTTP)

6.6.5 EXPERIMENT 2

After observing the high performance of the proposed CoAP-mobile host, we designed a test case to investigate the performance of a mobile phone and a Raspberry Pi model B+. We wanted to explore the feasibility of using a Raspberry Pi over the mobile phone in the mobile Web service provision.

First, we installed our CoAP Webserver on the Raspberry Pi model B+ and used a 4G dongle to connect to the internet. We measured the energy consumption of the device in idle state and when the server running under a different number of client requests and payloads. We applied the same setup 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 the **Error! Reference source not found.**), the mobile phone consumes less energy than the Raspberry Pi (RPi). With 20 concurrent client request (Figure 7. 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 behave differently because it consumes from 50 J to 60.9 J for the same amounts of payloads. There is a limitation we observed that we 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 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, we 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 7. Chart (a) & (b).

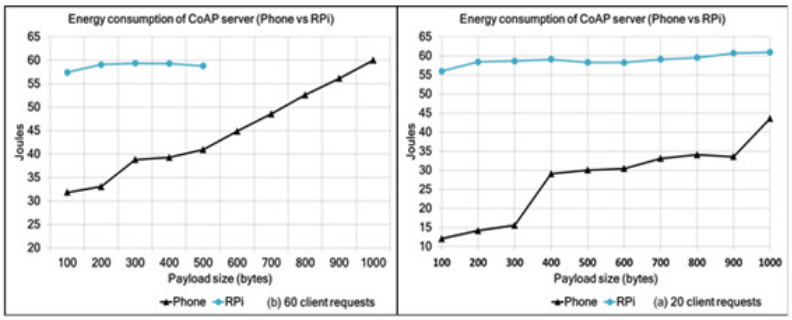

Figure 7. Comparison Of Energy Consumption (Phone Vs Raspberry Pi)

6.6.6 EXPERIMENT 3

In this experiment we added Global Sensing Service Scheduler into the mobile host. Then we measured the energy consumption for 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.

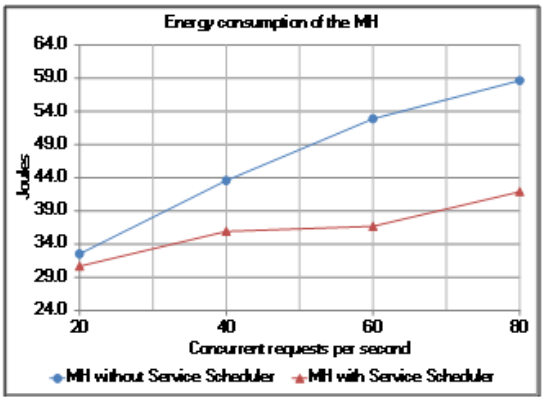

Figure 8. Comparison Of Energy Consumption

With the Sensing Service Scheduler, mobile host does not have to send a reply with the same data for all the clients, instead after sending 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 the **Error! Reference source not found.**. the Sensing Service Scheduler keeps minimizing 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 our work at [9].

7. CONCLUSION

In this paper, we have presented a lightweight mobile Web service provisioning framework for resource constrained environment. We extended our original work by adding service scheduling and the management of conflicts in order to improve the quality of service provisioning. Together with this improvement our mobile host enables mobile users to participate in different mobile phone sensing systems without affecting much of their hardware resources. During our

implementation and testing, we used REST and lightweight protocols in order to maintain the less complexity and 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 than the past traditional MWS frameworks.

As for the future research directions, we would like to improve the efficient service discovery features which are most important in the Machine-to-Machine communication.

ACKNOWLEDGEMENTS

This research is supported by the Estonian Science Foundation grant PUT360.

REFERENCES

[1] S. N. Srirama, M. Jarke, and W. Prinz, "Mobile web service provisioning," in Telecommunications, 2006. AICT-ICIW'06. International Conference on Internet and Web Applications and Services/Advanced International Conference on, 2006, p. 120.

[2] Z. Yun, G. Ren, and B. 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, 2007, pp. 6525–6528.

[3] I. K. Kim and J. H. Song, "Mobile Health reference architectures," in Information Society (i-Society), 2013 International Conference on, 2013, pp. 158–164.

[4] Z. Yu, D. Zhang, and D. Yang, "Participant Selection for Offline Event Marketing Leveraging Location-Based Social Networks."

[5] K. Mohamed and D. Wijesekera, "A lightweight framework for web services implementations on mobile devices," in Mobile Services (MS), 2012 IEEE First International Conference on, 2012, pp. 64–71.

[6] M. Jansen, O. Hordt, and J. Milatovic, "About the development of scenarios for mobile Web Service provisioning," in Computer and Information Technology (WCCIT), 2013 World Congress on, 2013, pp. 1–6.

[7] C. Chang, S. N. Srirama, and S. Ling, "An adaptive mediation framework for mobile P2P social content sharing," in Service-Oriented Computing, Springer, 2012, pp. 374–388.

[8] C. Chang, S. N. Srirama, and S. Ling, "SPiCa: a social private cloud computing application framework," in Proceedings of the 13th International Conference on Mobile and Ubiquitous Multimedia, 2014, pp. 30–39.

[9] C. Chang, S. N. Srirama, and M. 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, 2015.

[10] C. Chang, S. W. Loke, H. Dong, F. Salim, S. N. Srirama, M. Liyanage, and S. Ling, "An energy-efficient inter-organizational wireless sensor data collection framework," in Web Services (ICWS), 2015 IEEE International Conference on, 2015, pp. 639–646.

[11] M. Liyanage, C. Chang, and S. N. Srirama, "Lightweight Mobile Web Service Provisioning for Sensor Mediation," in Mobile Services (MS), 2015 IEEE International Conference on, 2015, pp. 57–64.

[12] Z. Shelby, B. Frank, and D. Sturek, "Constrained Application Protocol (CoAP) draft-shelby-core-coap-00," Online at http://tools. ietf. org/html/draft-shelby-core-coap-01, 2010.

[13] Bluetooth SIG, "Bluetooth Low Energy." .

[14] "Efficient XML Interchange (EXI) Format 1.0 (Second Edition)." [Online]. Available: http://www.w3.org/TR/2014/REC-exi-20140211/.

[15] S. Berger, S. McFaddin, C. Narayanaswami, and M. Raghunath, "Web services on mobile devices-implementation and experience," in Mobile Computing Systems and Applications, 2003. Proceedings. Fifth IEEE Workshop on, 2003, pp. 100–109.

[16] S. N. Srirama, M. Jarke, and W. Prinz, "Mobile Host: A Feasibility Analysis of Mobile Web Service Provisioning.," in UMICS, 2006.

[17] S. N. Srirama and M. Jarke, "Mobile hosts in enterprise service integration," Int. J. Web Eng. Technol., vol. 5, no. 2, pp. 187–213, 2009.
[18] R. K. Lomotey, Y. Chai, K. A. Ahmed, and R. Deters, "Web Services Mobile Application for Geographically Dispersed Crop Farmers," in Computational Science and Engineering (CSE), 2013 IEEE 16th International Conference on, 2013, pp. 151–158.
[19] R. K. Lomotey, S. Jamal, and R. Deters, "SOPHRA: a mobile web services hosting infrastructure in mHealth," in Mobile Services (MS), 2012 IEEE First International Conference on, 2012, pp. 88–95.
[20] S. Srirama and W. Prinz, "Concept, implementation and performance testing of a mobile Web Service provider for smart phones," phdthesis, Master's thesis, RWTH Aachen, Germany.(Cited on pages 2, 35, 36 and 45.), 2004.
[21] G. Gehlen and L. Pham, "Mobile web services for peer-to-peer applications," in Consumer Communications and Networking Conference, 2005. CCNC. 2005 Second IEEE, 2005, pp. 427–433.
[22] S. N. Srirama, "Publishing and discovery of mobile web services in peer to peer networks," arXiv Prepr. arXiv1007.2980, 2010.
[23] S. N. Srirama, M. Jarke, H. Zhu, and W. 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, 2008, pp. 668–674.
[24] Z. Ou, M. Song, H. Chen, and J. 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, 2008, pp. 195–200.
[25] Y.-S. Kim and K.-H. Lee, "A lightweight framework for mobile web services," Comput. Sci. Dev., vol. 24, no. 4, pp. 199–209, 2009.
[26] C. Chang, S. Ling, and S. Krishnaswamy, "Promws: Proactive mobile web service provision using context-awareness," in Pervasive Computing and Communications Workshops (PERCOM Workshops), 2011 IEEE International Conference on, 2011, pp. 69–74.
[27] S. N. Srirama and C. Paniagua, "Mobile web service provisioning and discovery in android days," in Proceedings of the 2013 IEEE Second International Conference on Mobile Services, 2013, pp. 15–22.
[28] C. van der Westhuizen and M. Coetzee, "A framework for provisioning restful services on mobile devices," in Adaptive Science and Technology (ICAST), 2013 International Conference on, 2013, pp. 1–7.
[29] G. Moritz, F. Golatowski, C. Lerche, and D. Timmermann, "Beyond 6LoWPAN: Web services in wireless sensor networks," Ind. Informatics, IEEE Trans., vol. 9, no. 4, pp. 1795–1805, 2013.
[30] C. Doukas, L. Capra, F. Antonelli, E. Jaupaj, A. Tamilin, and I. 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, 2015, pp. 192–197.
[31] S. N. Srirama, "MWSMF: A mediation framework for mobile hosts and enterprise on cloud," Int. J. Pervasive Comput. Commun., vol. 7, no. 4, pp. 316–338, 2011.
[32] A. P. Castellani, M. Gheda, N. Bui, M. Rossi, and M. Zorzi, "Web Services for the Internet of Things through CoAP and EXI," in Communications Workshops (ICC), 2011 IEEE International Conference on, 2011, pp. 1–6.
[33] Z. Shelby, "Embedded web services," IEEE Wirel. Commun., vol. 17, no. 6, p. 52, 2010.
[34] R. T. Fielding, "Architectural styles and the design of network-based software architectures," phdthesis, University of California, Irvine, 2000.
[35] F. AlShahwan and K. 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, 2010, pp. 174–179.
[38] S. N. Srirama and M. Liyanage, "Tcp hole punching approach to address devices in mobile networks," in 2014 2nd International Conference on Future Internet of Things and Cloud (FiCloud), 2014, pp. 90–97.