Implementing LWM2M in constrained IoT devices

Conference Paper · August 2015

DOI: 10.1109/ICWISE.2015.7380353

4 authors, including:

Vishwas Lakkundi
Freelance
20 PUBLICATIONS **91** CITATIONS

SEE PROFILE

Implementing LWM2M in Constrained IoT Devices

Suhas Rao, Devaiah Chendanda, Chetan Deshpande, Vishwas Lakkundi
Altiux Innovations
Bengaluru, India

***Abstract*—LWM2M is an emerging Open Mobile Alliance standard that defines a fast deployable client-server specification to provide various machine to machine services. It provides both efficient device management as well as security workflow for Internet of Things applications, making it especially suitable for use in constrained networks. However, most of the ongoing research activities on this topic focus on the server domain of LWM2M. Enabling relevant LWM2M functionalities on the client side is not only critical and important but challenging as well since these end-nodes are invariably resource constrained. In this paper, we address those issues by proposing the client-side architecture for LWM2M and its complete implementation framework carried out over Contiki-based IoT nodes. We also present a lightweight IoT protocol stack that incorporates the proposed LWM2M client engine architecture and its interfaces. Our implementation is based on the recently released OMA LWM2M v1.0 specification, and supports OMA, IPSO as well as third party objects. We employ a real world application scenario to validate its usability and effectiveness. The results obtained indicate that the memory footprint overheads incurred due to the introduction of LWM2M into the client side IoT protocol stack are around 6-9%, thus making this implementation framework very appealing to even Class 1 constrained device types.**

Keywords—Device Management; Internet of Things; LWM2M; Constrained Nodes; IoT Gateway; OMA Objects; IPSO Objects

I. INTRODUCTION

Lightweight M2M (LWM2M) is a system standard in the Open Mobile Alliance (OMA). It aims to develop a fast deployable client-server specification to provide machine to machine (M2M) services. It acts as an OMA device management (OMA-DM) successor for use in M2M and provides efficient device management as well as security workflow for IoT applications using the same protocol, thus offering enhanced simplicity. It also addresses service and management needs of constrained M2M devices over a number of transports and bearers. The LWM2M specification provides APIs for device configuration, connectivity monitoring/statistics, security, firmware update, server provisioning and so on. The widely used Constrained Application Protocol (CoAP) provides in-built binding for LWM2M, thus making it particularly appealing for the Internet of Things (IoT).

A. State-of-the-Art

Most of the articles on this topic currently available in literature address LWM2M from the server perspective, mainly focusing on efficient management of billions of things using the cloud infrastructure, gateways and mobile devices.

Reference [1] presents an implementation of a programmable and low-cost IoT gateway in an embedded system which has been used in monitoring systems in an IoT test bed. It also discusses a protocol to translate different sensor data into a uniform format and server assisted provisioning methods. It defines the use of CoAP as a transmission protocol instead of HTTP and provides a multi-protocol gateway architecture. The oneM2M architecture which integrates various other device management frameworks and M2M network management issues are also addressed. It also covers IP-based network management protocols and ongoing work on self-management.

Reference [2] addresses self-management of M2M device and its configuration, M2M service enablement for end user and device management along with remote entity management such as dynamic software updates and so on, all from a server perspective.

References [3] and [4] mainly describe a smart M2M gateway-based architecture to manage large volumes of M2M devices and cover the internal structure of the gateway and APIs to manage M2M devices and endpoints. They illustrate a gateway managing connected devices in a smart home scenario. Their architectures depict the gateway as a collection of web services and how to configure and manage mobile clients such as smart phones and tablets running applications of connect and control type.

In this paper, we define the overall protocol stack architecture for the end-device domain. We also propose a LWM2M client engine architecture framework for use in IoT end-nodes based on an open standards, namely OMA LWM2M [5] and IPSO for standard data formats for sensor objects [9]. Hence our work helps advance the state-of-the-art by providing an efficient architecture for the end-device domain, which enables the clients to be managed more efficiently by employing industry standard protocols and methods.

Our proposal is designed to achieve the following key goals to ease the application development and integration of third party platform-specific components:

- Compliance with OMA and IETF standards
- Objects compliant with OMA and IPSO standards
- Well defined APIs
- Compact, modular and flexible architecture
- Easily configurable
- Minimal device management functionality from application
- Easy to port and integrate
- Optimized for integration with constrained devices

This paper is organized as follows: Section II provides an overview of the OMA LWM2M standard, while Section III presents the design and architectural aspects of our proposal in detail. Section IV covers the implementation phase including client and server setups in addition to providing performance metrics. It also illustrates a real-world application scenario. Finally, conclusions are drawn in Section V.

II. LWM2M STANDARD

LWM2M provides a light and compact secure communication interface along with an efficient data model, which together enables device management and service enablement for M2M devices. As with other device management standards such as OMA DM, the Lightweight M2M solution is called an Enabler.

The OMA Lightweight M2M Enabler, consisting of a LWM2M Client (M2M device) and a LWM2M Server (M2M service/platform/application), employs a client-server architecture plus CoAP with UDP/SMS transport bindings as shown in Fig. 1. The enabler is targeted, in particular, at constrained devices, e.g. devices with low-power microcontrollers and small amounts of Flash and RAM over networks requiring efficient bandwidth usage. At the same time, LWM2M can also be utilized with more powerful embedded devices that benefit from efficient communication.

The LWM2M Enabler defines the application layer communication protocol between a server and a client. The LWM2M Server is typically located in a private or public data centre and can be hosted by the M2M Service Provider, Network Service Provider or Application Service Provider. The LWM2M Client resides on the device and is typically integrated as a software library or a built-in function of a module or device. Following four logical interfaces are defined between the server and client:

1. Bootstrap: allows LWM2M Bootstrap Server to manage the keying, access control and configuration of a device to enrol with a LWM2M Server.

2. Device Discovery and Registration: allows an LWM2M Client device let the LWM2M Server know its existence and register its capability.

3. Device Management and Service Enablement: allows the LWM2M Server to perform device management and M2M service enablement by sending operations to the Client and to get corresponding responses from the LWM2M Client.

4. Information Reporting: allows the LWM2M Client to report resource information to the LWM2M Server; can be triggered periodically or by events.

The LWM2M communication model is based on simple COAP methods such as GET, PUT, POST, and DELETE with bindings over UDP or SMS as transport layer. The binary encoded message overheads will only be a few bytes and the flat, simple objects with uniform URI across devices makes the protocol best suited for constrained device connectivity and easy management.

The LWM2M Enabler defines a simple resource model where each piece of information made available by the LWM2M Client is a Resource. The Resources are further logically organized into Objects. The LWM2M Client can have any number of Resources, each of which belongs to an Object. For example the Firmware Object contains all the Resources used for firmware update purposes.

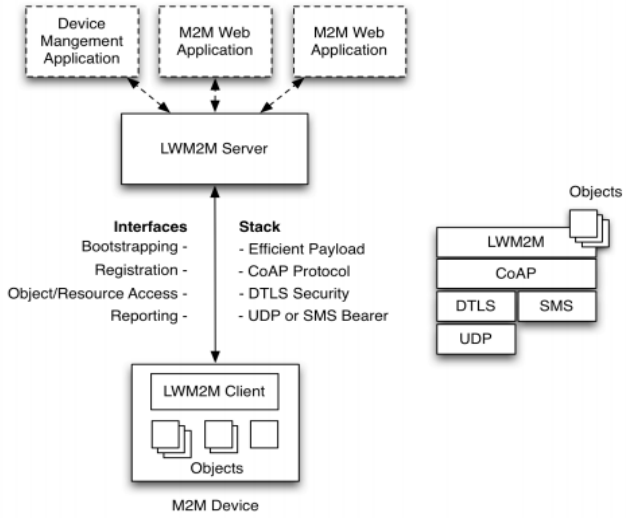

Fig. 1. LWM2M architecture [5].

All the LWM2M interfaces along with corresponding call-flows between a client and a server are illustrated in Fig. 2.

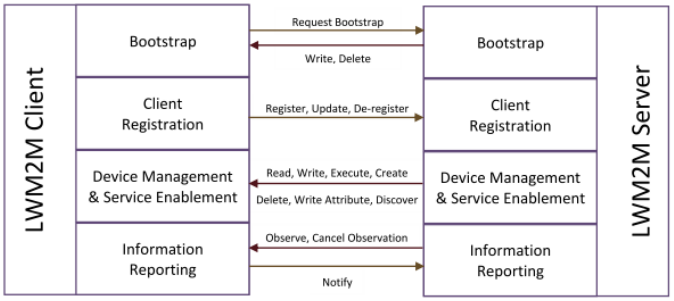

Fig. 2. LWM2M interfaces and workflow.

The LWM2M interfaces use CoAP as an underlying transfer protocol across IP and SMS bearers. For realization of these interfaces, only the basic binary CoAP message header, and a small subset of options are required as described earlier.

CoAP supports a URI in requests and in LWM2M only path segment and query string URI components are needed. The URI path is used to simply identify the interface, Object Instance or Resource that the request is for, and is encoded in URI-Path option. The LWM2M Registration interface also makes use of query string parameters to pass on meta-data with the request separately from the payload. Each query parameter is encoded in a URI-Query option. Likewise, the LWM2M operations for each interface are mapped to CoAP Methods [7]. Every LWM2M operation except Notify must be a Confirmable CoAP message and Notify can be either a Confirmable or Non-Confirmable message when UDP is used.

III. DESIGN AND ARCHITECTURE

A. IoT Connectivity Protocol Stack

The proposed lightweight IoT connectivity protocol stack is illustrated in Fig. 3. It consists of the following layers:

- LWM2M Protocol Engine Core: implements the complete OMA device management and service enablement operations and provides the application programming interfaces (API's).
- Interface Layer: implements and provides a well-defined generic API's to access various interface layers such as OMA/IPSO Object the underlying connectivity layer (CoAP) and platform specific implementations such as Firmware over The Air (FoTA) from the LWM2M client protocol.
- CoAP: defines the message header, request/response codes, message options, and retransmission mechanisms; LWM2M uses only a subset of features defined by CoAP.
- DTLS: provides the security channel between LWM2M Server and LWM2M Client for all the message exchanges.
- UDP/SMS: currently the UDP binding for LWM2M is mandatory, whereas SMS binding is optional.
- IPv6/6LoWPAN: provides IPv6 support to constrained devices via 6LoWPAN.
- IEEE 802.15.4 MAC/PHY: provided by the chip vendor and hence are not in our direct control.

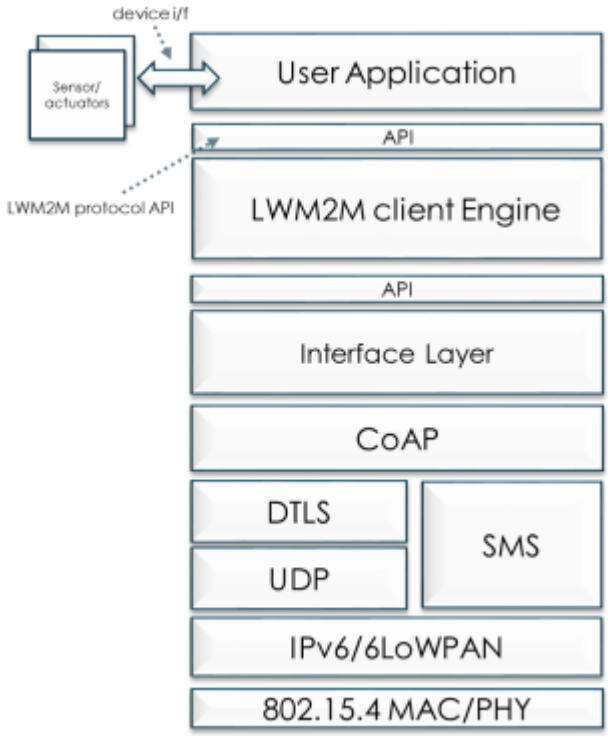

Fig. 3. Proposed lightweight protocol stack architecture.

B. LWM2M Client Engine

The architecture of LWM2M client engine is shown in Fig. 4. The availability of interface layer makes the LWM2M client engine modular, compact and easily deployable across various platforms with minor or no changes.

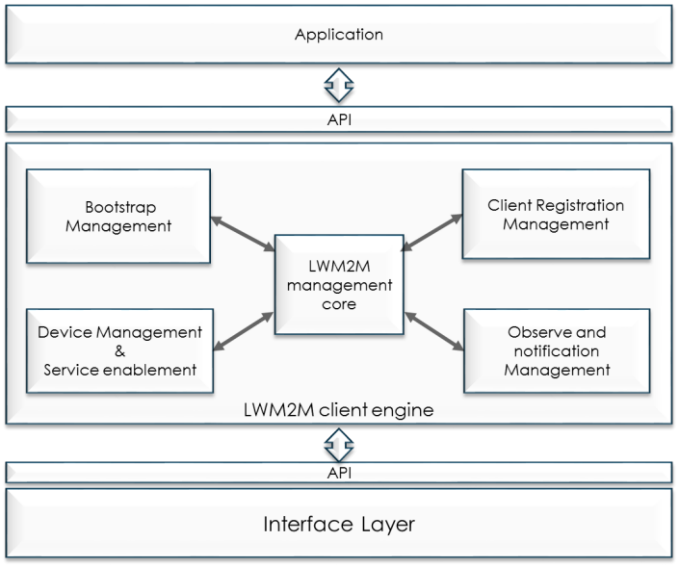

Fig. 4. LWM2M client engine architecture.

The components of the LWM2M client engine and their functionalities are outlined as follows:

- LWM2M Management Core: implements the necessary core functionalities, by interconnecting and managing other protocol interface layers within LWM2M client engine. It is also responsible for creating required Objects, initializing resource values from the configuration parameters available for the given application/platform requirements. The configuration can also come from device specific bootstrapping interfaces.
- Client Registration Management Module: implements the registration, automatic registration update and de-registration procedures with the configured LWM2M server(s).
- Observe and Notification Module: implements the automatic notification of resource value to the observing LWM2M server(s) as defined in [10].
- Device Management & Service Enablement Module: implements the device management functionalities and service enablement interfaces to enable operations on objects and resources defined by OMA LWM2M specification such as create, delete, read, write, execute, discover etc.
- Bootstrap Management Module: implements the interfaces and procedures needed to support various bootstrapping mechanisms as per LWM2M protocol specification.

C. Interface Layer

The following modules are part of interface layer which shall be used by the LWM2M client engine.

- OMA/IPSO Object: implements the growing list of OMA, IPSO and third party registered standard object definitions and provides the interfaces to update/modify resource(s) value by the registered LWM2M server as well as device interfaces.
- Connectivity Interface Layer: provides interfaces to use the underlying CoAP protocol implementation available for the given platform.
- Firmware Update Module: implements the firmware update procedure as defined by the LWM2M specification to handle the request from the registered LWM2M server and provides the interfaces which can make use of platform specific firmware update implementations.
- Object/Device Configuration: supports selective configuration and use of application specific Objects from the vast pool of OMA/IPSO objects available.

D. LWM2M Client Interface APIs

The LWM2M client engine provides the following interfaces:

- North-side Interface: to enable access to LWM2M protocol by the user application. This interface layer implements APIs to be used by the application to enable LWM2M client functionality on the device.
- South-side Interface: to enable integration with the underlying platform-dependent modules such as CoAP protocol stack, object configuration, FoTA and so on.

E. LWM2M Objects and Resources

The following OMA defined standard objects and a sample IPSO defined objects are available as part of our Interface Layer. This list is expandable based on availability from OMA/IPSO and is also configurable based on the application requirements.

1. LWM2M Security Object: provides the keying material of a LWM2M Client appropriate to access a specified LWM2M Server and a LWM2M Bootstrap Server. These LWM2M Object Resources must only be changed by a LWM2M Bootstrap Server or Bootstrap from Smartcard and must not be accessible by any other LWM2M Server.
2. LWM2M Server Object: provides data related to a LWM2M Server.
3. Device Object: provides a range of device related information that can be queried by the LWM2M Server, and a device reboot and factory reset function.
4. Firmware Object: enables management of firmware that is to be updated. This object includes installing firmware package, updating firmware, and performing actions after updating the firmware.
5. IPSO object: a sample IPSO defined object such as IPSO actuator object [9].

F. LWM2M Test Compliance

The LWM2M client protocol stack is tested against the LWM2M client interoperability test cases and may include the optional features defined by the OMA LWM2M enabler test specification (ETS) version 1.0 [6].

The LWM2M enabler tests are carried out using the LWM2M protocol and objects, in addition to employing the underlying protocols such as CoAP.

G. Benefits of Proposed LWM2M Client Engine

The proposed LWM2M client engine (enabler) for our connectivity middleware stack is designed to achieve the following key goals to ease the application development and the integration of third party platform-specific components.

- Protocol compliant with OMA and IETF standards.
- Objects compliant with OMA LWM2M standard.
- Well defined APIs.
- Compact, modular and flexible architecture: it implements only the core interfaces defined by the standard and allows application developers to choose underlying platform-specific connectivity stack as well as configure the stack components through a flexible and modular approach.
- Easily configurable: objects and device parameters are easily configurable to suit different application needs.
- Minimal device management functionality from application: most of the standard device management functionalities are implemented as part of the core LWM2M client engine, adhering to LWM2M standard and hence the application developer is free from implementing and/or managing complex device management functionalities.
- Easily portable and ease of integration: since platform dependent modules are abstracted through the interface layer, the core LWM2M client engine is easily deployable on various platforms with minor or no changes.
- Better suited for integration with constrained devices: with only minor changes required to the interface layer components in order to make use of platform-specific modules, seamless integration into the new platform is achieved.

IV. IMPLEMENTATION AND PERFORMANCE ANALYSIS

A. Environment Setup

1) LWM2M Client Setup

We use Texas Instruments' CC2538 System-on-Chip (SOC) [11] as the LWM2M client device. It is a wireless microcontroller for high-performance WPAN applications. It combines a powerful ARM Cortex-M3-based MCU system with 32 KB on-chip RAM and 512 KB on-chip flash with a robust IEEE 802.15.4 radio running at 32 MHz clock, enabling the device to handle complex network stacks that support security, demanding applications as well as over-the-air download. The client runs Contiki OS with the LWM2M stack running directly over RFC-compliant COAP layer [8]. The callbacks for LWM2M stack are registered through the already available REST interface of Contiki [12]. The client node connects to the network through the border router that also uses CC2538 SOC running RPL border router application [13] running on Contiki OS. The border router is connected to the PC host through the USB-to-Serial interface. IP packets are transferred through Serial Line Internet Protocol (SLIP), so the PC host is expected to be running an open source SLIP utility called *tunslip* [14]. On receiving COAP packets, the PC host transfers them to the destination over the Internet.

2) LWM2M Server Setup

Leshan [15] is an open-source OMA LWM2M server and client implementation. In our setup, we use the server implementation of Leshan to test our LWM2M client. Leshan comes packaged as a simple Java executable that can be invoked like any other *jar* file. Once the Leshan is up and running, one can connect on Leshan UI from any standard web browser using *http://<Server-IP_addr>:8080*. Leshan provides a very simple UI to get the list of connected clients and interact with client resources. LWM2M clients can now be registered with the Leshan LWM2M server. Leshan UI also offers a simple web interface to manage the LWM2M client device.

3) LWM2M Objects

Our LWM2M client has been tested with the following four standard LWM2M objects and one IPSO actuation object:

- Security object
- Device object
- Server object
- Firmware object
- Standard IPSO-defined smart actuator object

4) OMA LWM2M DevKit

The OMA LWM2M *DevKit* [16] is an add-on for the Mozilla Firefox Web browser. It adds support for the OMA LWM2M M2M protocol and enables manual interaction with a LWM2M Server directly from the Web browser. This way, developers and users can interactively explore and comprehend the new protocol for machine-to-machine communication. This has been used to evaluate and compare the behavior of our implementation of the LWM2M client.

B. Performance Metrics

Performance metrics of our implementation in terms of memory footprints (code and data) are presented in Table I.

TABLE I. STACK MEMORY FOOTPRINTS

Memory Footprint in bytes		
Component	***Code Size (Flash)***	***Data Size (RAM)***
LWM2M alone	8764	820
CoAP alone	9724	1385
Full Stack + Contiki OS	86121	12292

It is clearly evident from Table I that introducing the LWM2M protocol layer into our IoT stack results in an additional memory footprint of just around 9% flash memory and 6% RAM. Even these additional memory footprints would easily fit into Class 1 (~10 KiB RAM and ~100 KiB Flash [17]) constrained devices that are often employed in IoT applications. In addition, the average flash memory required to support a LWM2M object is around 700 bytes. Considering the huge benefits that LWM2M brings to the IoT stack in terms of device management and M2M service enablement, these overheads are not burdensome at all.

To elaborate further, the memory footprints of different constituent sections of LWM2M are shown in Table II.

TABLE II. LWM2M MEMORY FOOTPRINTS

Memory Footprint in bytes		
LWM2M Component	***Code Size (Flash)***	***Data Size (RAM)***
LWM2M Client Core and APIs	2260	400
Registration Management	1804	350
Object Mapping and Management	1920	24
Observe-Notify Management	552	16
TLV Format	2228	30

C. Application Scenario

LWM2M as a device management protocol layer can be used to manage and control various functionalities of a device. In order to demonstrate a real world application scenario, we use it to manage a light bulb remotely over the network.

A light resource was connected to the smart actuator of the LWM2M client device and the client device was registered with the LWM2M server - Leshan. We were able to read the status of the light resource, in this case a bulb and were successfully able to control the bulb remotely through the web interface provided by the Leshan server.

Our goal was to control the light bulb from a remote destination as illustrated in Fig. 5. In order to achieve this, a bulb was connected to the smart-actuator that in turn is controlled by the TI CC2538 SOC on which the LWM2M client is running. This client is connected to the border router over IEEE 802.15.4 radio.

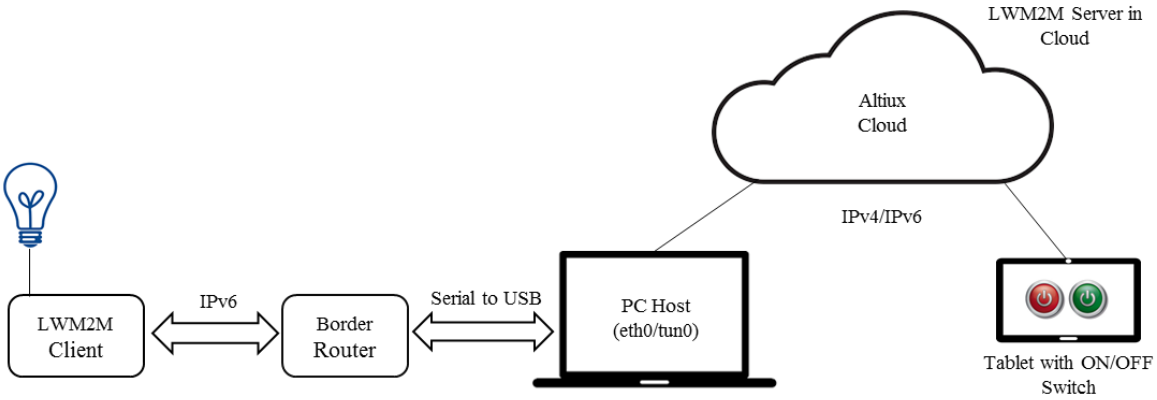

Fig. 5. LWM2M application scenario – remote control of light bulb.

In our scenario, the border router is a CC2538 SOC running out of the box RPL border router Contiki application [13]. It is connected to the external network through the *tunslip* [14] interface running on a standard Ubuntu PC, which in turn is connected to the Leshan LWM2M server hosted in the cloud. Once the LWM2M client registers itself with the LWM2M server, it is possible to control the bulb through the web user interface hosted by the LWM2M server itself or a standalone tablet on a standard 8080 TCP port.

In addition to managing the light bulb, we were also able to continuously monitor the status of the bulb when LWM2M server made an *Observe* request to the LWM2M client. In this case, whenever the light was switched ON or OFF the LWM2M server was automatically notified by the client regarding the change of event and the corresponding status was updated on the web interface as well.

V. Concluding Remarks

In this industry-track paper, we have addressed the core challenges faced by developers and practitioners while implementing LWM2M in constrained IoT devices. We have proposed the LWM2M client-side architecture and its complete implementation framework carried out over Contiki-based IoT end-nodes. This paper also presented a lightweight IoT protocol stack that incorporates the proposed LWM2M client engine architecture and its interfaces. Our implementation is based on the recently released OMA LWM2M v1.0 specification, and supports OMA, IPSO as well as third party objects and resources.

To validate the veracity of our proposal, we employed a real world application scenario and performed analysis of the results obtained, which indicate that the memory footprint overheads incurred due to the introduction of the LWM2M protocol layer on the client side protocol stack are no more than 6-9%, thus rendering our design proposal and implementation framework not only lightweight but practical as well for use in constrained IoT devices.

References

[1] Z. Sheng et al, "Lightweight Management of Resource Constrained Sensor Devices in Internet-of-Things," IEEE Internet of Things Journal, April 2015.

[2] S. Datta and C. Bonnet, "A Lightweight Framework for Efficient M2M Device Management in oneM2M Architecture," in Proc. IEEE 10th International Conference on Intelligent Sensors, Sensor Networks and Information Processing ISSNIP 2015, April 2015, Singapore.

[3] S. Datta and C. Bonnet, "Smart M2M Gateway based Architecture for M2M Device and Endpoint Management," in Proc. IEEE International Conference on Internet of Things ITHINGS 2014, September 2014, Taipei, Taiwan.

[4] A. Sehgal et.al, "Management of resource constrained devices in the internet of things," IEEE Communications Magazine, vol.50, no.12, pp.144,149, December 2012.

[5] Open Mobile Alliance (OMA), "LightweightM2M Technical Specification v1.0," November 2014, available online at www.openmobilealliance.org.

[6] Open Mobile Alliance (OMA), "Lightweight Machine to Machine Enabler Test Specification v1.0," February 2014, available online at www.openmobilealliance.org.

[7] G. Klas, F. Rodermund, Z. Shelby, S. Akhouri and J. Höller, "Lightweight M2M: Enabling Device Management and Applications for the Internet of Things," White Paper, February 2014, available online at http://community.arm.com/docs/DOC-8284.

[8] The Constrained Application Protocol (CoAP), IETF RFC 7252, June 2014, available online at https://tools.ietf.org/html/rfc7252.

[9] IPSO smart objects version 1.0, available online at http://www.ipso-alliance.org/smart-object-guidelines.

[10] K. Hartke, "Observing Resources in CoAP", draft-ietf-core-observe-16, work in progress, December 2014.

[11] TI CC2538 - Wireless Microcontroller System-On-Chip for 2.4 GHz IEEE 802.15.4, 6LoWPAN, and ZigBee Applications, available online at http://www.ti.com/product/cc2538.

[12] Contiki - the open source OS for the Internet of Things; the official git repository available online at http://www.contiki-os.org.

[13] RPL border router example on Contiki OS, available online at https://github.com/contiki-os/contiki/tree/master/examples/ipv6/rpl-border-router.

[14] Contiki's Tunslip utility, available online at https://github.com/contiki-os/contiki/blob/master/tools/tunslip6.c.

[15] Leshan - an OMA LWM2M implementation in Java, available online at https://github.com/eclipse/leshan.

[16] OMA LWM2M DevKit extension for Mozilla Firefox, available online at https://addons.mozilla.org/en-US/firefox/addon/oma-lwm2m-devkit/.

[17] Terminology for Constrained-Node Networks, IETF RFC 7228, available online at https://tools.ietf.org/html/rfc7228.