Enhancing Lightweight M2M Operations for Managing IoT Gateways

Bilhanan Silverajan*, Mert Ocak†, Jaime Jiménez†, Antti Kolehmainen*
*Tampere University of Technology, Finland
{bilhanan.silverajan, antti.kolehmainen}@tut.fi
†Ericsson Research, Finland
{mert.ocak, jaime.jimenez}@ericsson.com

***Abstract*—Heterogeneity in constrained networks and gateways is perhaps one of the single greatest challenges facing end-to-end management of devices and networks in the IoT. Today, the LightWeight M2M (LWM2M) protocol, leveraged on open Internet standards, has become a strong contender for REST-based IoT management. However, significant challenges exist for remote gateway management, particularly for proxies and exposing proprietary or vendor-specific operations. This paper presents some effective solutions in overcoming these challenges in LWM2M using object modelling and linking. The presented approaches are also validated with prototypes.**

I. Introduction

Several major Standards Development Organisations (SDOs) have undertaken considerable effort in establishing protocols, architectures and communication guidelines in the Internet of Things (IoT) domain. The standardisation of the Constrained Application Protocol (CoAP) [1] was an important milestone, effectively extending the Representational State Transfer (REST) architecture, the basis for predominant web API model today [2], towards constrained IoT devices and smart objects. CoAP also is becoming the foundation for REST-based device management in IoT. The Open Mobile Alliance (OMA) developed the Lightweight M2M (LWM2M) architecture [3] around CoAP, specifying guidelines, interfaces and data models for managing devices, sensors and actuators. A similar strategy was adopted by the IPSO Alliance [4], which augmented LWM2M with semantically interoperable object models as building blocks which can be composed together to describe more complex smart objects.

Because the growth of the IoT is fairly organic, there would always be a need to integrate and interconnect a variety of existing and deployed smart objects, many of which use legacy and non-IP communication using different kinds of gateways. Even with natively IP-enabled smart objects in the future, connecting billions of devices in the IoT would encompass multiple Layer 2 and Layer 3 solutions. This would require IoT gateways to not only supply connectivity and routing, but also behave as protocol translators or application level proxies.

We characterise an IoT gateway according to its functionality. It can supply wireless IP connectivity to devices and sensors by bridging two kinds of Layer 3 networks with fixed or cellular uplinks. It can proxy between non-IP networks and the Internet. It can facilitate endpoint reachability for management operations in the presence of firewalls and NATs using packet encapsulation or tunnelling. It can finally provide application-level architectural interoperability among existing management tools and systems.

Other ways of categorising IoT gateways in terms of the computational power, form factor or energy consumption patterns are also possible. Quite often, gateways in IoT can fulfill multiple roles. Many IoT gateways used in constrained environments also possess similar properties as IoT devices and smart objects in terms of hardware heterogeneity, computational ability, battery lifetimes or remote management. Consequently, as long as any potential limitations are addressed, managing IoT gateways using IoT device management protocols and architectures would, on the surface, appear to be a logical step for the configuration, control, and lifecycle management procedures. Even in situations where the gateways themselves are not constrained or are more computationally powerful, there are significant advantages in terms of semantic interoperability and evolution, when a management server can adopt the same methods in managing an IoT device as well as the gateways along the network path. This forms the basis of our work described here.

In this paper, we apply our experience gained with LWM2M-based management and standardisation activities in IPSO towards utilising LWM2M and IPSO data models for effectively managing IoT gateways. While in this section, we showed the importance of gateways and gateway management in the IoT, section II provides an overview of the LWM2M architecture and IPSO Objects. Section III offers a deeper insight into the major challenges of using LWM2M for gateway management particularly with exposing dynamic operations as well as proprietary and non-IP data models. Section IV offers some novel solutions for overcoming these limitations, by using new object models as well as web linking and the Information Reporting Interface of LWM2M. To add further clarification in this paper, we present a simple example of how this can be achieved in practice, by demonstrating an example energy management scenario in Section V. Findings that validate our work are discussed in Section VI before conclusions are presented in Section VII. Related work is discussed throughout the paper as concepts are presented.

II. LWM2M Architecture and IPSO Objects

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 [6]. The architecture defines a client-server communication model. A LWM2M Server is typically located in a private or public data centre and can be hosted by the M2M service provider to manage LWM2M clients. 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. An optional Bootstrap Server can be used to manage the initial configuration parameters of LWM2M Clients during bootstrapping the device.

Management operations between the server and client are grouped logically into four interfaces: The Bootstrap Interface, the Device Discovery and Registration Interface, the Device Management and Service Enablement Interface, and the Information Reporting Interface. As mentioned previously, LWM2M uses CoAP for communication as the underlying transport protocol. CoAP itself has similar method calls and response codes as the Hypertext Transfer Protocol (HTTP). LWM2Ms management operations map onto CoAP method calls and response codes. CoAP resources are also identified using Uniform Resource Identifiers (URIs). LWM2M introduces a data model where LWM2M resources are logically collected into LWM2M Objects. This is achieved by introducing a simple hierarchical object model, in which objects represent sensor or actuator types, with a list of resources representing the properties specific to that object type.

The OMA also maintains a naming authority called the Open Mobile Naming Authority (OMNA). The OMNA operates an Object and Resource Registry with which objects and resources can be standardised with a name, description, types and so on. An unambiguous identifier is also assigned. For on-the-wire efficiency, this ID is used instead of names, during CoAP operations. Multiple objects of the same type can be differentiated by their Instance ID. This allows objects and resources to be mapped in a standard way into CoAP resources and URI path components, in the form of `/<object ID>/<object instance ID>/<resource ID>`.

It is common that at startup, each LWM2M Client registers several distinct objects to a LWM2M Server. Upon registration, the Server assigns a unique endpoint ID to each Client. After registration, the Server is further able to differentiate various endpoints by prepending the endpoint ID to the object and resource identifiers. Subsequently, the Server is able to perform read, write and execute management operations on Client objects and resources using standard CoAP method calls and URIs. Such a CoAP URI might be `coap://lwm2m.example.org/123/3/0/2` which retrieves the serial number (Resource ID 2) of object instance 0 of the LWM2M Device object located at the LWM2M endpoint hosted on lwm2m.example.org having an endpoint ID of 123.

In addition to the OMA, the IPSO Alliance also defines smart objects and object models. IPSO Objects are defined in such a way that they do not depend on the use of CoAP, and any RESTful protocol is sufficient. Nevertheless, to develop a complete and interoperable solution the IPSO Object model is based on LWM2M specification and object model [7]. While LWM2M uses objects with fixed mandatory resources, IPSO Objects use a more reusable design. As devices increase in complexity, IPSO specifies how to compose sophisticated Objects by using simpler Objects as building blocks and linking them with specific resources called Object Links. Object Links point to other objects within the Client and are expressed as two concatenated integers separated by a colon, such as 3304:2, where the first integer represents the Object ID and the second the Object Instance. IPSO Objects are also registered with the OMNA.

III. Considerations of Gateway Management in IoT

A study undertaken by the Internet Architecture Board (IAB) to understand the impact of IoT on existing Internet infrastructure led to the publication of an informational standard on architectural considerations for smart object networking [5]. One of the findings outlined is the device-to-gateway communication pattern, which is an architectural pattern connecting proprietary and non-IP nodes to the Internet using gateways. While the pattern considers reachability and end-to-end communication, such a communication pattern also impacts gateway management, operations and the management of non-IP endpoints. These are described in the subsections below.

A. Integrating non-IP and proprietary data models

The use of LWM2M and IPSO data models between the LWM2M server and managed endpoints requires both reachability and end-to-end IP connectivity. The specifications do not cater for the management of end-points which use short range wireless radios such as Bluetooth Low Energy (BLE), Z-Wave or Zigbee.

One approach towards managing non-IP nodes is to masquerade the nodes' properties in the gateway itself. Should a gateway be used to bridge these networks into the IoT, it can also translate between the kinds of data models expected by the LWM2M architecture and the proprietary data models used by the end-points. Integrating such proprietary models to the network requires the gateway to serve as an intermediary on behalf of these end-points in terms of registration and management operations. Often this can be an effective solution for exposing properties as a proxy to the LWM2M server in a RESTful way as standard LWM2M or IPSO Objects. This is highly effective for managing very resource-constrained endpoints, providing semantic interoperability for connected end devices. It can also be used to expose proprietary gateway models for configuration and control.

As an example, the Unified Configuration Interface (UCI) [8] of the well-known OpenWRT Linux platform for gateways organises configuration information as various files with well-defined names in */etc/config/*, such as *network*, *firewall*, *system*

Resources in IPSO Gateway System Object		
Name	ID	Access Type
Hostname	0	R,W
Timezone	1	R,W
DNS Servers	2	R,W
NTP Servers	3	R,W

```
In /etc/config/system:

config system
        option hostname 'IoT-TUT-GW'
        option zonename 'Europe/Helsinki'
        option dns '8.8.8.8'
        option cronloglevel '8'

config timeserver 'ntp'
      list server '0.openwrt.pool.ntp.org'
      list server '1.openwrt.pool.ntp.org'
```

Fig. 1. Excerpts showing mapping between IPSO and UCI system data models

and *wireless*. Reading these files would reveal that internally, UCI internally models configuration-specific data into well-defined sections called *zones* with nested properties called *options*. Correlation between the nested UCI configuration data and LWM2M or IPSO hierarchical data models can therefore be performed. Figure 1 depicts this correlation between the system configuration data used by UCI and the IPSO Gateway System Object. The IPSO Gateway System Object can be used directly with LWM2M, although, at the time of this writing, its Object ID as well as URN are still subject to change and final assignment by IPSO.

A similar strategy can be employed to model the properties of non-IP devices and sensors. In BLE, data exchanged through the Generic Attribute Profile (GATT) is organised hierarchically into *Services* and *Characteristics*. A single service is a logically grouped collection of characteristics, while characteristics contain type, values and properties such as supported operations. GATT services and characteristics are therefore very similar to LWM2M Objects and Resources. Consequently, an intermediate gateway can be used to manage BLE endpoints in LWM2M using this approach. Integrating such proprietary models to the network requires the gateway to translate between the data models.

However, such translation is done using proprietary methods in most of the current gateway implementations and hence, creates silos between different gateway manufacturers. As an example, Bluetooth SIG publishes GAP and GATT REST API white papers [9] [10] to standardize the APIs defined on Bluetooth gateways but common data models to seamlessly integrate the Web and BLE data models are needed to provide interoperability between these two technologies. The BIPSO project, on the other hand, aims at harmonising BLE data models with those of IPSO, by providing a standard mapping between IPSO Objects to BLE Characteristics with well-defined Characteristic Values [11].

The LWM2M architecture currently does not also possess adequate semantics for describing such proxying, as while CoAP itself supports proxies via the *Proxy-URI* option, how this can be achieved and defined in LWM2M remains open. Consequently, while the gateway can integrate and host data models on behalf of proprietary or non-IP endpoints, it needs to masquerade the objects and resources of each endpoint on its LAN as its own. In other words, the management server remains unaware that such Objects refer to physically distinct endpoints external to the gateway. Ideally, discovering and using proxy functionality should be incorporated into future data models which unambiguously assert the location and type of managed endpoints residing behind such a gateway. This way, the runtime interaction would become expressive enough to cleanly separate end-to-end management from any underlying connectivity issues.

B. Exposing proprietary and gateway-specific operations

IoT device management rarely takes into account non-atomic, transaction-based or system-specific method calls which are inherently present in many operations related to gateway configuration. Often, these calls also differ from vendor to vendor. While the LWM2M data and operation models support read, write and execute operations on a resource (effectively a CoAP GET, PUT or POST operation), a gateways proprietary or native configuration API often needs to be invoked upon a change in a resources representation. This is effectively an implicit process not exposed to the management server as the object model does not attach any semantics to the kinds of native operations that could or should be invoked upon actions on Objects and resources. When applied to gateway management this can become an issue, as the object model does not accurately capture the kinds of operations and configuration changes a gateway may need to perform in the background, to properly reflect the resource state in the LWM2M object model.

As an example, consider a packet filtering mechanism on an access point, for which an example LWM2M or IPSO object model comprises of firewall objects. Each firewall object represents a single rule, while each resource represents an option, such as a name, source or destination address, incoming interface, connection state. In order to first properly construct and then trigger the firewall rule on the gateway as well as to subsequently monitor its status, each action on the object to be taken needs to be modeled as a resource. Moreover, an additional final commit resource is needed to trigger the firewall rule to make the changes effective. Without any additional semantics, similar kinds of object and resource models need to be developed for other policy objects, system configuration and the gateways network interfaces. A different gateway implementation may also choose to perform its runtime firewall management in a different manner, and incorporate additional resources to reflect these operations in a static manner within a different object model. In the long run, this leads to untenable and fragmented Object models with poor interoperability. Clearly, a better solution would be to retain a core reusable Object model representing the properties of a gateway or an endpoint being proxied by the gateway, and conceive a mechanism which exposes system-specific operations dynamically at runtime.

In HTTP-based REST models, resources can be dynamic supplied from a server to a client with hypermedia by embedding a REST constraint called Hypermedia As The Engine of Application State (HATEOAS) [12]. Using HATEOAS, an origin server only exposes resources directly necessary

for the current context, while returning possible operations on the object or resource as hyperlinks and relation types implementing standard REST verbs. While using HATEOAS increases the messages exchanged between a client and a server, it allows the responses from the server to the client to be more transparent, and allows run-time discovery of allowed operations on the server by a client.

If HATEOAS can be used, the firewall object can be simplified so that the gateways native API invocations are exposed to a management server over a REST interface when necessary. A commit hyperlink could be exposed under a name resource if and only if the firewall object has yet to be deployed, for example. Hypermedia-based APIs, content types and HATEOAS, however are currently unsupported with LWM2M.

IV. Strategies for IoT Gateway Management

In order to overcome the challenges outlined in Section III, our solutions were inspired by a technique employed by IPSO. Recognising that attempting to define complex devices as discrete Objects can easily result in large and unsustainable models, IPSO deliberately defined collections of Objects as simple building blocks. These are then aggregated by developers to construct Composite Objects using Web Linking [13]. Such a Composite Object would have resource lists similar to simple Objects. However they would also contain Resources whose types are defined as LWM2M Object links. These would then contain reference URIs which are mapped to other Object Instances assembled to create the Composite Object Model, as outlined in section 4 of [7].

A. Semantics for Proxy Management

The issues described in Section III-A can be overcome using a similar approach as IPSO by using Web Linking. This requires the Object model of a managed non-IP endpoint to furnish enough semantics to the LWM2M Server which describes and identifies the gateway being used as a proxy. For example, we have drafted 2 IPSO Objects which aim at working with low-powered non-IP nodes, particularly for BLE. These are depicted in Fig 2 as the Personal Area Network (PAN) Interface Object, and the BLE Generic Attribute Profile (GATT) Object.

The PAN Interface Object is a generic Composite Object that can be used for modelling multiple PAN interfaces of either an endpoint or a gateway. For each interface, the *Type* Resource specifies the kind of PAN this interface represents, such as BLE, Zigbee, Classic Bluetooth and so on.

The address associated with the interface is also expressed, followed by 2 Resources representing the properties of its communicating peer. For an endpoint, these would represent the gateway's PAN address as well as the LWM2M Endpoint ID supplied by the Server, while for a gateway, multiple instances of the *Peer Address* Resource would represent managed end-points, and optionally, any Endpoint ID. In the case of a BLE-based endpoint, multiple instances of the *Low Power Interface* Resource would be linked to multiple instances of the BLE GATT Object, with each BLE GATT Object representing a single BLE service.

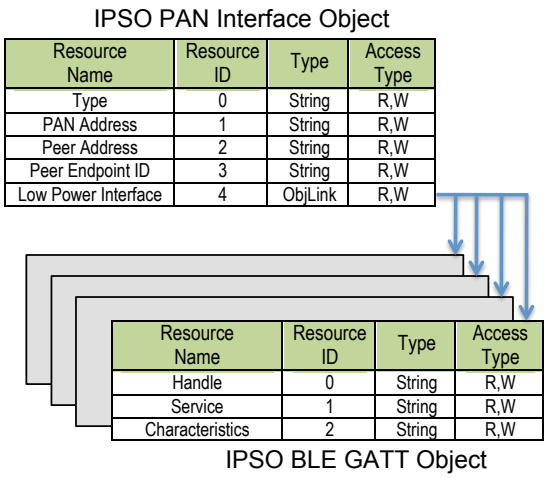

Fig. 2. IPSO PAN and BLE GATT objects.

Describing managed endpoints via a proxy in this manner aupplies an LWM2M Server with enough structured information to correlate as well as understand the kinds of non-IP endpoints being managed and through which gateways communication is possible.

B. Interaction model with dynamic linking

Gateways often have system-specific operations that need to be performed during management. Section III-B outlined the current shortcomings that prevent LWM2M Servers from understanding and invoking them. However, just as with proprietary data models from the previous section, a simple but effective technique using Object links allows an LWM2M Server to be aware of contextual operation invocation on the gateway, based on the current representational state of the gateway's objects and resources. Additionally, the LWM2M Information Reporting Interface is relied upon as a notification mechanism at runtime to dynamically supply information of gateway-specific operations to the LWM2M Server. This notification mechanism is based on CoAP Observe [14], which allows an observer to register an interest on an observable CoAP resource, so that when the representational state of the resource changes, the observer is notified.

Classic LWM2M and IPSO data model depict Resources and Objects in a fairly static structure. To this, another class of gateway objects need to be introduced, which we term Interaction Objects. Interaction Objects contain a list of executable Resources, each of which represents a system-specific operation required by the gateway, in order to effect runtime changes when requested by the LWM2M Server. Such an instance of the Interaction Object could be linked to a specific Gateway Object Instance together with another Interaction Object having observable Resources, which computes the state of the Object Instance and updates its Resource values to supply the object link to the correct Interaction Object

(A) IPSO Operations Object

Resource Name	Resource ID	Type	Access Type	Observ-able?
Op List	0	ObjLink	R	Yes
Active Op	1	Integer	R	Yes

(B) IPSO BLE Commands Object

Resource Name	Resource ID	Type	Access Type
HCI_TOOL	0	String	X
HCI_CONFIG	1	String	X
GATT_TOOL	2	String	X

(C) IPSO UCI Commands Object

Resource Name	Resource ID	Type	Access Type
UCI_GET	0	String	X
UCI_SET	1	String	X
UCI_COMMIT	2	String	X

Fig. 3. IPSO Interaction Objects.

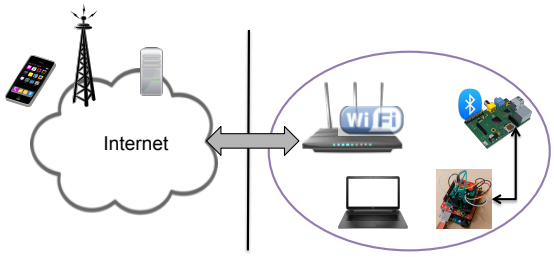

Fig. 4. Experimental setup

TABLE I
IPSO WIRELESS GATEWAY INTERFACE OBJECT

Resource Name	Resource ID	Type	Access Type
Ifname	0	String	R,W
Mode	1	String	R,W
Disabled	2	Boolean	R,W
SSID	3	String	R,W
MAC Address	4	String	R,W
IP Address List	5	String	R,W
Transmission Power	6	Integer	R,W
Network	7	String	R,W

Instance and the Resource ID representing the system-specific operation) to be invoked.

Fig 3 illustrates three draft IPSO Objects that, when composed together with an IPSO or LWM2M Gateway object, can accurately provide the management server with hypermedia-like operations on the gateway. For example, assume that the gateway registers an IPSO Wireless Interface Gateway Object, composited together with the IPSO Operations Interaction Object as well as the IPSO UCI Commands Interaction Object. The Wireless Interface Gateway Object is shown in Table I. At registration, the *OP List* Resource in the Operations Object would point to the UCI Commands Object Instance registered by the gateway, while the *Active OP* Resource value would reflect the currently active operation. In this particular case, the value would be 0, to reflect that the currently active operation would be the *UCI_GET* executable resource. If the LWM2M Server, at some future point in time, choose to disable the Wireless Interface Object by setting the value of the *Disabled* Resource to "1", the value of *Active OP* would be changed by the gateway to "1", reflecting that the next active operation on the Wireless Interface Gateway Object would be the *UCI_SET* executable resource. Subsequently, if the LWM2M Server has a previously established notification relationship with the Resources in the IPSO Operations Object, it would be automatically informed by the gateway of a pending system-specific operation.

V. EVALUATION AND TESTING

The enhancements to LWM2M discussed in the previous section were evaluated by implementing the proposed Objects for IoT gateways in an experimental setting. Our main use case for the validation of our approach was a simple energy management use case. Our test environment consisted of several devices to mimic a heterogenous gateway management scenario using LWM2M. This is depicted in Figure 4. Two kinds of gateways were set up. The first was TP-Link AC-1200 Wi-Fi Access Point (AP) connected to the Internet over a fixed Ethernet connection offering IPv4/IPv6 access. The stock firmware was replaced with the OpenWRT Chaos Calmer distribution and additional hardware and software was installed so that the AP was capable of scanning and communicating over BLE as a GATT client. The second gateway was a Raspberry Pi 2 serving as a BLE gateway primarily for bridging and connecting to BLE sensors. The BLE gateway was connected to the AP as a Wi-Fi client. Both gateways had embedded C-based LWM2M client code from the Eclipse Wakaama project modified and implementing the IPSO Wireless and System Gateway Objects, proposed PAN Object, BLE GATT Object as well as the Interaction Objects. A custom GATT server was also implemented in the BLE gateway with a proprietary service, enabling authorised clients to connect and write to a specific service and characteristic to control its Wi-Fi interface. Additionally, the gateway periodically announced its presence by transmitting BLE advertisements. The precise power being supplied to the BLE gateway was non-invasively monitored in real-time using a data acquisition (DAQ) device which provided real-time, precise, energy consumption statistics of the BLE gateway. The DAQ device was built in-house, containing an energy consumption monitoring toolkit, a Hall-Effect current sensor connected to an Arduino board and controlled over a Raspberry Pi Model B. Finally an Ubuntu laptop was also added into the setup for monitoring purposes. The laptop was also connected to the WiFi network and was capable of BLE-based communication. All 3 devices (the AP, the BLE gateway and the laptop) used the Linux BlueZ Bluetooth stack.

Both gateways registered to, and were managed over the cloud, using a public Java-based LWM2M server from the Eclipse Leshan project. The LWM2M server was hosted by the Eclipse Foundation and was residing at *leshan.eclipse.org*. An HTTP interface to the Leshan server allowed control, configuration and management operations using a web browser. This was performed using an Android-based phone forming a connection to *https://leshan.eclipse.org* over LTE.

To perform our tests, we initially controlled the server to successfully retrieve and write resources on both gateways. This was performed over IP to both gateways. However, both the AP as well as the BLE gateway registered their PAN

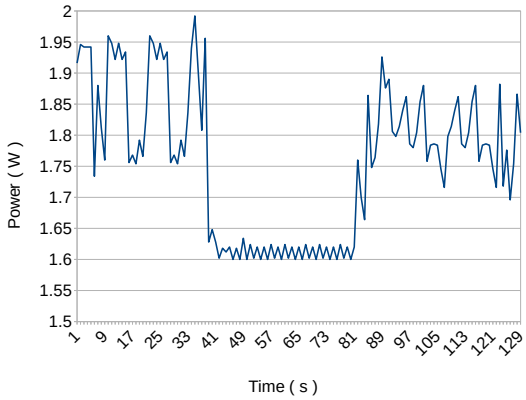

Fig. 5. BLE Gateway power consumption

Objects, with registration updates being performed every 300s. Both gateways also listed each other as peers in their respective BLE GATT Object Instances. Our intention was to see if using our approach with exposing system specific operations dynamically as well as proxying and working with non-IP endpoints, we can first successfully power down the Wi-Fi interface of the BLE gateway, thereby losing direct Internet connectivity between the LWM2M Server and the BLE Gateway without adversely affecting management. Subsequently, we wanted to demonstrate that by using the AP as a proxy, the LWM2M Server would be able to use the BLE Command Object of the AP to communicate with the BLE Gateway and power its Wi-Fi interface up once more. During these steps, the power consumption of the BLE Gateway should be observed to see if we were successful.

VI. Findings

Using the object modelling techniques described in the previous sections, the LWM2M server successfully executed several kinds of operations on the BLE gateway, depending on the communication context. The server was able to retrieve or change resource representations on the BLE gateway directly when IP connectivity was present, and the BLE gateway was able to provide the correct object links to allow the server to disable the WiFi interface. Additionally, once the WiFi interface was deemed to be disabled, the server was also able to retrieve the correct endpoint identifier of the AP and utilise it as a proxy to enable the WiFi interface via the APs BLE interface. The BLE gateway then resumed communication with the server over the higher powered WiFi interface. These events are graphically depicted in Fig 5, which shows the power consumption of the BLE gateway during this test scenario. From the graph, it can clearly be seen that periodic transmissions over Wi-Fi resulted in bursts of energy consumption, and when the Wi-Fi interface was powered down at around 40s, the power consumption of the BLE gateway decreased significantly. In order to allow the BLE gateway to update its LWM2M registrations, Wi-Fi was once again powered up and the subsequent activities such as association with the AP, obtaining IP addresses, resuming services and updating LWM2M registrations over Wi-Fi, are once again clearly visible. In addition to the power measurements, these activities were also verified by using the laptop to solicit ICMP responses from the BLE gateway to Ping packets, inspecting the LWM2M serverfor fresh registrations and subsequent updates from the BLE gateway at the Leshan server.

VII. Conclusion

Work is ongoing for REST-based management of IoT devices and gateways, particularly in the IETF. RESTCONF [15] is one such example. RESTCONF is hoever, HTTP-based, and it faces limitations for constrained device management. Ongoing work on CoAP Management Interface (CoMI) [16] aims to adapt RESTCONF for use in constrained environments using CoAP. Additionally, the use of HATEOAS as well as Web Linking to define the relation types between object instances on different endpoints would heavily aid IoT gateway management. These are actively being pursued in other SDOs.

In this paper, we presented novel methods to manage IoT gateways effectively, using LWM2M. We focused on two specific areas, namely allowing vendor-specific and proprietary gateway operations to be effectively modelled as resources in an object model which the management server can interact with, and the ability to expose and manage endpoints by explicitly modelling proxy operations that gateways may possess. These were achieved by extending existing object models to contextually expose permitted operations on a gateway to a management server at runtime.

References

[1] Z. Shelby, K. Hartke, and C. Bormann. "Constrained Application Protocol (CoAP)", IETF 7252, June 2014.
[2] W. Jung, S. I. Kim and H. S. Kim, "Ontology modeling for REST Open APIs and web service mash-up method," Information Networking (ICOIN), 2013 International Conference on , vol., no., pp.523,528, 28-30 Jan. 2013.
[3] Open Mobile Alliance, "Lightweight Machine-to-Machine Technical Specification v1.0, Candidate Enabler", August 2016.
[4] IP for Smart Objects, "IPSO Objects", http://ipso-alliance.github.io/pub/, Accessed September 2016.
[5] H. Tschofenig, J. Arkko, D. Thaler, and D. McPherson, "Architectural Considerations in Smart Object Networking", IETF RFC 7452, March 2015.
[6] J. Prado,"OMA Lighweight M2M Resource Model", IAB IoT Semantic Interoperability Workshop 2016, March 2016
[7] J. Jimenez, M. Koster and H. Tschofenig, "IPSO Smart Objects", IAB IoT Semantic Interoperability Workshop 2016, March 2016
[8] OpenWrt UCI System, https://wiki.openwrt.org/doc/uci, Accessed September 2016.
[9] Bluetooth SIG, "GAP REST API White Paper", April 2014.
[10] Bluetooth SIG, "GATT REST API White Paper", April 2014.
[11] BIPSO, http://bluetoother.github.io/bipso/, Accessed September 2016.
[12] M. Kovatsch, Y. N. Hassan, and K. Hartke. "Semantic Interoperability Requires Self-describing Interaction Models", IAB IoT Semantic Interoperability Workshop 2016, March 2016
[13] M. Nottingham, "Web Linking", IETF RFC 5988, September 2016.
[14] K. Hartke, "Observing Resources in the Constrained Application Protocol (CoAP)", IETF RFC 7641, September 2015.
[15] A. Bierman, M.Bjorklund, and K. Watsen, "RESTCONF Protocol", ID-ietf-netconf-restconf, September 2016.
[16] P. van der Stok, and A. Bierman, "CoAP Management Interface", ID-vanderstok-core-comi, March 2016.