Extensions to LwM2M for Intermittent Connectivity and Improved Efficiency

Abdulkadir Karaagac, Matthias Van Eeghem, Jen Rossey, Bart Moons, Eli De Poorter, Jeroen Hoebeke
Ghent University - imec, IDLab, Department of Information Technology
Ghent, Belgium
Email: abdulkadir.karaagac@ugent.be, jeroen.hoebeke@ugent.be

***Abstract*—In order to extend the Internet technologies into constrained networks, there have been several research efforts and innovations over the past few years. These efforts resulted in several networking standards and protocols which have led to the emergence of the Internet of Things (IoT). Within this enormous ecosystem of protocols, the Lightweight Machine-to-Machine (LwM2M), a standard for device and service management, is a very promising candidate to achieve global interoperability, especially when constrained devices are involved. In this paper, we propose extensions to the LwM2M in order to improve communication efficiency and introduce Intermittent Connectivity, which will improve support for Low-Power Wide Area Networks (LPWANs) in LwM2M. For this purpose, we introduce two new object models, namely, Notify and Batch. The Notify Object enables the creation of reverse interaction models and Ready-to-Receive (RTR) functionality, which allows LwM2M clients to send periodic updates of resources without any need for a request and to notify LwM2M servers that it is ready to receive downlink messages and to keep the network connectivity open if it is necessary for downlink communication. Finally, the Batch object allows LwM2M servers to perform actions on multiple resources in a device via a single request.**

***Index Terms*—IoT, CoAP, LwM2M, LPWAN, Batch**

I. Introduction

As a result of the latest advances in the Internet of Things (IoT) technologies, a wide range of new possibilities and applications emerge with a significant number of devices (sensors, actuators, vehicles, wearables, implants etc.) being connected to the Internet: 30 billion devices by 2020 [1]. The majority of these intelligent *things* are envisioned to be constrained in terms of processing capability, memory and energy as well as to have low unit price and long battery life. Due to these limitations, widely-used Internet protocols are not appropriate for constrained devices. Therefore, a new set of open protocols and standards were designed to be used by these devices in order to integrate with the rest of the Internet.

In this context, several standards and novel wireless technologies (e.g. LPWANs) have been proposed in order to achieve energy and resource efficient communication for constrained devices. Under the lead of Internet Engineering Task Force (IETF), many working groups have been formed for the realization of IP-based connectivity for constrained devices [2], [3], [4]. Moreover, several research and standardization initiatives are gathered and constituted to target interoperability and M2M understandability in the IoT. For instance, oneM2M [5], Open Mobile Alliance (OMA) [6] and Internet Protocol for Smart Objects (IPSO) Alliance [7] are leading global organizations that deliver specifications and architecture for efficient Machine-to-Machine (M2M) communication and global interoperability for the future of the IoT. The combination and coordination of these protocols creates a standardized way for integrating these constrained devices into the Internet, which makes them an important enabler of the IoT.

Lightweight M2M (LwM2M) is a device and service management protocol from Open Mobile Alliance (OMA) and offers series of fundamental functionalities for IoT devices including secure bootstrapping, resource and device discovery, efficient transfer of management and application data. According to OMA LWM2M specification, all the interactions (read, write etc.) are initiated by the LWM2M server. Therefore, a client can only start sending notifications for certain resources only upon observation requests for those resources. This feature creates a limitation for several IoT solutions where the application logic on the IoT device being programmed to periodically send an update of a set of resource values or where there is a sensor data to be collected from devices in LPWAN Networks where the nodes can be in sleep mode for a long period of time and for Network Address Translation (NAT) connected devices.

Therefore, in this paper, we propose extensions to LwM2M protocol in order to overcome these issues and also improve the communication efficiency. First of all, we introduce the *Ready-to-Receive (RTR)* functionality in LwM2M by means of the *Notify* object, which can improve support for Networks with Sleepy or NAT connected Devices. The Notify Object can also be used to create *reverse interaction model* to get periodic updates of resources without any need for a request. In addition, we also create the ability to write to and read from a subset of resources of an object or multiple resources over different objects using a single request with a newly proposed object: the *Batch*. Also theoretical and practical analyses are provided in order to demonstrate the contribution of Batch to achieve more efficient communication in LwM2M.

The remainder of the paper is organized as follows. Section II provides a detailed background about the LwM2M protocol. In Section III, the LWM2M extensions for Intermittent Connectivity are described. And in Section IV, the Batch object is introduced in details with theoretical and practical analysis about its contribution in the network performances. Finally, Section V concludes the paper.

II. The LwM2M Protocol

LwM2M is an efficient and secure client-server protocol, from the Open Mobile Alliance (OMA), with several functionalities for managing resource constrained IoT devices [8]. LwM2M defines efficient interactions for remote application management via several interfaces built on top of the CoAP protocol, which is a web transfer protocol for resource constrained devices and networks, standardized by IETF CoRE working group [9]. The overview of the typical LwM2M interaction model and the structure of a LwM2M object model is presented in Figure 1.

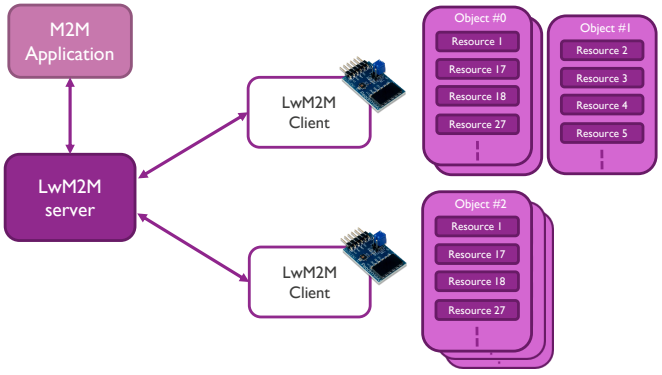

Fig. 1. An overview of LwM2M APIs and Data Models.

The LwM2M protocol relies on uniform object models which are collections of mandatory and optional resources representing atomic pieces of information. These object models are sets of pre-structured URI templates and registries of object identifiers with semantics attached. This provides machine-readable representations of the semantics.

LwM2M clients hold multiple objects, each with a unique identifier: the object ID. Some objects are mandatory (e.g. LwM2M Security) and some are optional. Each object can have zero or more instances. Every instance shares the same set of resources but the exact values in these resources can differ from instance to instance. Each resource within these objects has a Resource ID (RID), which uniquely defines the resource. Referencing a specific resource is done using the Uniform Resource Identifier (URI) */object_id/instance_id/resource_id* and */object_id/instance_id* is used to reference a specific object instance.

The LwM2M defines resources with different access types (Read, Write, Read/Write, Execute) and with a wide range of data types, such as String, Integer, Float along with homogeneous lists of these types [8]. But it is not possible to mix types within a single list. The standard supports a number of data formats; TLV (Type-Length-Value), JSON, Plain Text and binary data formats.

III. Intermittent Connectivity in LwM2M

Since the LwM2M specification only defines interactions which are initiated by the LWM2M server, client cannot instantiate any communication (after bootstrapping and registration). Due to this limitation, for the devices which are only reachable for a certain amount of time after each uplink communication (NAT, Firewalls, port filtering etc.) and for the devices that are reachable for downlink communication only after an uplink packet (LPWAN Networks), as illustrated in Figure 2, the LwM2M protocol needs further consideration. Otherwise the LwM2M server will not be able to reach to the client. A concrete example could be a sensor sending a value per hour and sleeping for the remaining time.

In order to overcome these issues, we propose extensions to LwM2M protocol and we introduce the *Ready-to-Receive* functionality by means of a new object: *Notify*. This functionality allows clients to send a notification to a predefined URI, to inform the server that they are awake for downlink messages and/or keep the connection open for downlink messages. Then already buffered or notification-upon created messages/requests can be transmitted upon the notification. Also a NAT-connected device can stay connected for downlink requests if the notification periodicity can be adjusted according to the NAT session timeout.

The details about the Notify object is provided in Table I. *Resource URI* represents a Web Link to the URI of the resource or instance that the updates will include. *SSID* represents the resource defining which external server that the update should be sent to. While, *URI Path* is the exact path on the server to write the update to using a PUT operation. The payload of this update message is the value of the resource with defined in *Resource URI*. Further, *periodicity* defines how often an notification should be sent and *Send Confirmable Message* lets the user define which type (confirmable, non-confirmable) of CoAP message should be sent. Finally, *No Response* is used to make the server not to send a response to specific (or all) types of messages as described in [10]. In this model (also for following sections), the IDs for the proposed objects (2346, 2347, 5913 etc.) are not standardized, but have been selected among the unassigned IDs according to the LwM2M Object and Resource Registry [11]. However, a final ID assignment should go via registration procedure.

TABLE I
The Notify Object (ID: 2347).

RID	Name	R/W	Type	Description
5915	Resource URI	R/W	String (Link)	Link to the URI of the target resource.
5916	SSID	R/W	Integer	Short Server ID.
5917	URI Path	R/W	String	URI path on the server.
5918	Periodicity	R/W	Integer	Period of uplink transmissions (sec).
5919	Confirmable	R/W	Boolean	Set CoAP messages to be Confirmable.
5920	No-Response	R/W	Boolean	Enable No-Response option in message.

The Notify object is also offering functionality to create *reverse interaction model*, which allows a user/server to register on a device to periodically receive updates for resources defined in *Resource URI*. This creates a solution for several IoT solutions where the application logic on the IoT device being programmed to periodically send an update of a set of resource values to a configured or pre-defined server address. Such a mechanism can also save an important amount of resources especially in case of several clients (thousands), as investigated in Section IV-C3.

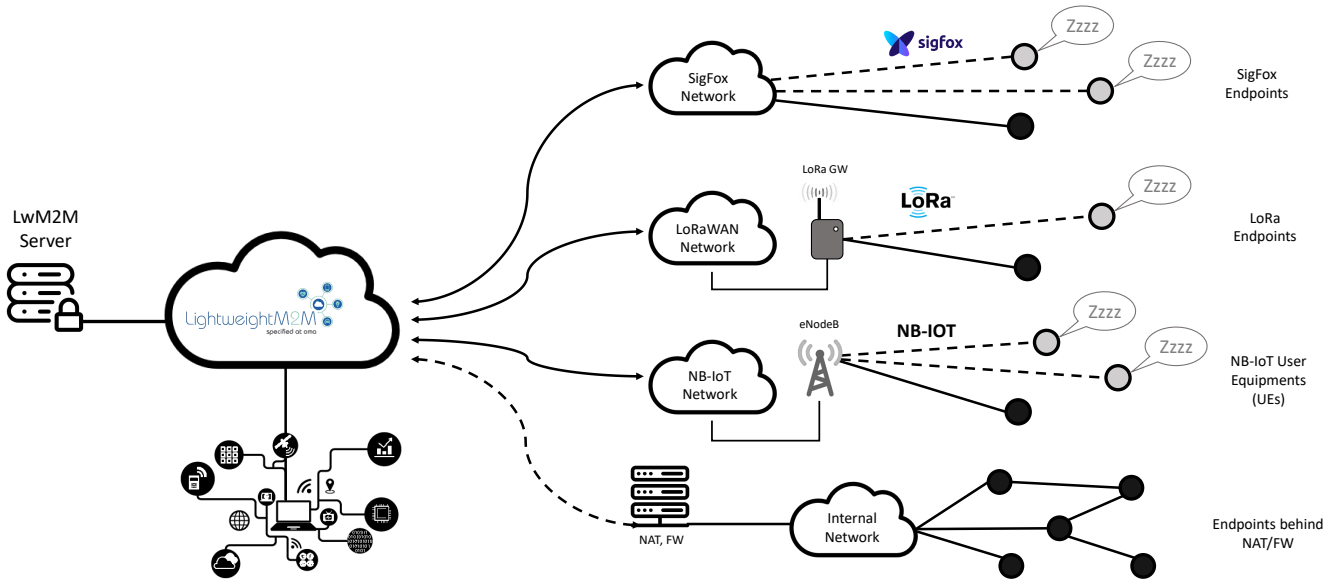

Fig. 2. The Intermittent Connectivity for LPWAN Networks and Networks behind NAT/FW.

IV. The Batch

Although the LwM2M protocol is providing efficient interactions for transferring service and application data, there are still some limitations. Currently, LwM2M allows reading an entire object instance or a certain resource by using a single request, however it is not possible to read a subset of resources of an object or multiple resources over different objects using a single request. So, separate requests have to be sent in order to read the values of multiple resources. Similarly, it only allows an object instance to be updated partially or entirely using a single request. This means that it is possible to edit multiple resources simultaneously as long as they belong to the same object instance. As is the case with reading, it is not possible to write to multiple resources across object instances using a single request.

In this context, IPSO defines *Composite Objects* where multiple objects can be combined using the Web Linking Framework [12]. However, this only provides semantic information about the objects which are linked, but it does not allow aggregating related or non-related objects or a subset of resources in a single request or response. Alternatively, recently proposed CoAP methods (FETCH, PATCH and iPATCH) [13], which enable to access and update only a certain part of a resource, improve the efficiency compared to primary CoAP methods. However they do not allow operations on multiple resources across different objects and they introduce a level of overhead due to added target resource URIs in the requests.

A. The LwM2M Batch Object

The Batch object is a new custom LwM2M object that allows the user to perform actions on multiple resources in a device by sending a single request. The overall Batch object is presented in Table II with two mandatory resources, named *Batch Configuration* and *Batch Value*.

TABLE II
The Batch Object (ID: 2346).

RID	Name	R/W	Type	Description
5913	Batch Configuration	R/W	String (List)	The list of aggregated resources.
5914	Batch Value	R/W	String (List)	Values of the resources defined in Batch configuration.

The Batch Configuration (5913) resource allows users to specify the resources that they are interested in, whereas the Batch Value (5914) resource allows the user to read from or write to the resources defined in the Batch Configuration. Both of these resources have type of 'list of strings' and are both readable and writable. If the aggregated resource are integers, floating point numbers or booleans, their encoding in the Batch Value would not be efficient at the moment, but a more efficient encoding (e.g cbor) for the value resource of the Batch object can be studied.

B. The Batch Operation

An example flow of the Batch create, read and write operations is visualized in Figure 3. In this example, the Batch configuration is first set to contain two resources: */x/y/z* and */a/b/c*. The second step simply reads what was just written. The configuration and value resources are directly linked with each other: the first string in the Batch value resource corresponds to the first URI in the Batch configuration etc. In the example, the value 42 corresponds to the resource */x/y/z*, and "test message" is the value of resource */a/b/c*. In the third step, the Batch value resource is read. The values of the resources in the Batch configuration are collected into a list and returned. Those values are collected into a list of strings and returned in a single request. The fourth step shows the operation to edit the values of multiple underlying resources at a time.

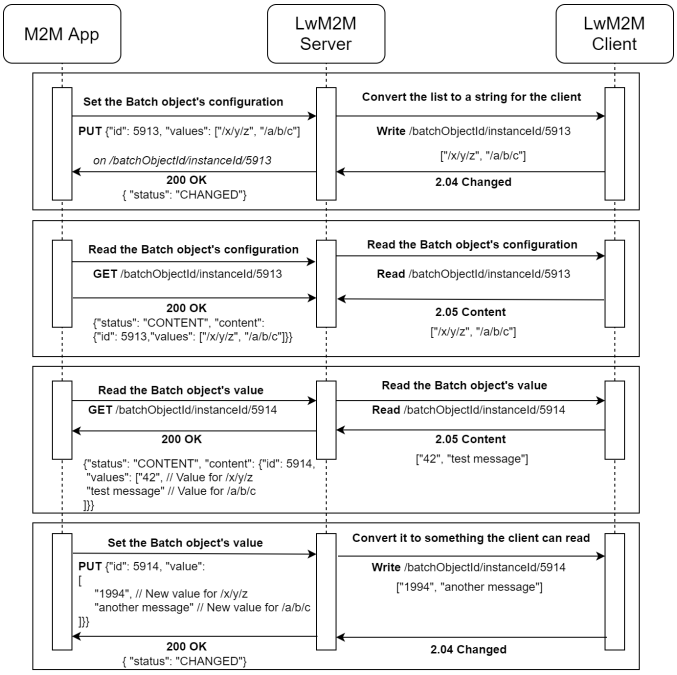

Fig. 3. A sequence diagram showing the interaction with the Batch object.

Since each operation on batch object applies to each of the aggregated resources, Read-only and Write-only resources cannot be aggregated into a single batch. Therefore, there needs to be a validation process for the content of the batch during creation.

When using the Batch object, updating multiple resources is an atomic operation. Either all resources of the batch are updated or none are, e.g. in case the packet is lost for example. However, using individual requests to update multiple resources in or across objects, the data in the resources might be partially updated in case of the loss or delay of a subset of packets, and therefore left in an invalid state. With Batch, one can ensure that all necessary resources are updated jointly which will ensure that the object is not left in an invalid state.

The Batch object also supports CoAP Observe functionality, which allows CoAP clients to observe resources on CoAP servers. If a value of a resource is changed through the Batch object, the Batch object makes sure to propagate this change to all of the potential observers of this resource. The reverse is done as well: if a resource is present in the Batch configuration and it is changed externally (i.e. not through the Batch object), observers of the Batch object are notified of a change as well.

C. Evaluation: The Contribution of the Batch

In this section, we performed theoretical and practical analysis of various communication scenarios involving LwM2M devices in order to investigate the potential contribution of the Batch approach. For this purpose, we compared the communication performance (required data to exchange, latency) for operations on several resources by means of first the Batch, which aggregates all the resources, and then multiple separate requests.

1) Theoretical Analysis: As it can be expected, the Batch approach is resulting in aggregated payload with larger packet sizes. However, due to the smaller number of requests and responses, the Batch object can provide a lot more efficient operations in terms of the total number of bytes to be exchanged. In order to demonstrate this efficiency improvement, we defined a simple scenario where a LwM2M server performs Read requests on different number of resources (each holds a response with an information of 4 Bytes long) in a LwM2M client. The amount of data (bytes) to be exchanged in order to perform these Read operations is provided in Figure 4 for varying number of resources to be read in TLV and JSON encoding schemes. This figure shows the fact that the Batch operation can improve the communication efficiency drastically compared to the case of using multiple requests. However, it is important to emphasize that the improvement of the Batch operation would change based on the number, type and length of the resources to be aggregated.

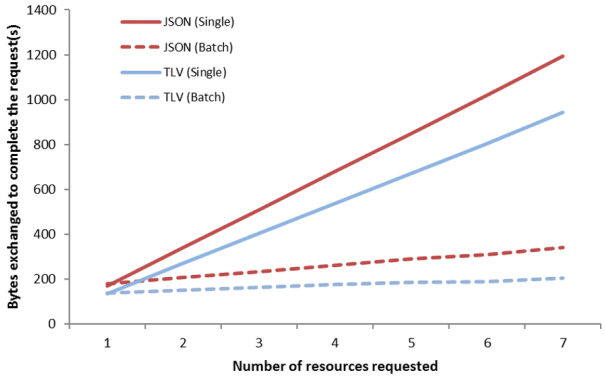

Fig. 4. The amount of bytes required to complete multiple requests.

Similarly, as the number of packets to be exchanged is decreased with the use of Batch, the total latency for certain operations will also diminish. In order to demonstrate this, we performed a simple analysis for the use of the Batch operation in a bandwidth-constrained LPWAN technology, using LoRa as a suitable example [14]. In a LoRA network, the total latency for a number of packet exchange would result in the total latency of all requests and responses along with the delay coming from the duty cycle regulations and a constant value:

$$latency = \sum_{i=1}^{n} airtime(req_n) + airtime(resp_n) + D(a_n) + C \quad (1)$$

In this equation, the airtime represents the packet transmission time and it depends mostly on the spreading factor (SF7-SF12), coding rate and bandwidth used [15]. According to the LoRA airtime calculator [16], 1 single request consists of 16 bytes CoAP header and 11 bytes payload, which adds up to 27 bytes packet length, resulting in 82 ms of airtime. Requesting 6 resources separately would thus result in 492 ms of sending time. Aggregating 6 of these requests in a single Batch, would require a single 16 bytes CoAP header and 66 bytes of payload. These 82 bytes result in 164 ms of airtime, or 3 times less airtime for the same amount of data. So, the

Batch object decreases the number of request and response messages, so it decreases the total airtime of the LoRa packets. Secondly, $D(a_n)$ represents the delay which would come from the duty cycle regulations (e.g. 1% in EU 868) and it is directly proportional to packet airtime. Therefore, it will also decrease as the Batch operation decreases the total communication airtime. The constant (C) part consists of the processing delay and the time to transfer packets between the gateway and the cloud and this constant delay will add up and be larger for the sending multiple requests.

However, combining multiple resources will not always result in a gain in efficiency. Whenever the size of the aggregated response message is larger than the maximum transfer unit (MTU) allowed to be sent over the network, the message will be split into multiple packets. This is expected to impact the communication performance negatively. This now poses a trade-off: using single requests requires a separate packet to be sent for every resource, but from a certain number of resources, the Batch object's packet size will exceed the MTU and it will also require multiple packets using block transfer. This second option could cause the Batch object to become less efficient.

2) Practical Evaluation - Exposing Device Location: In order to see how well this idea stacked up against the state of the art, the Batch object and related functionalities are implemented for both a LwM2M client and server. For the LwM2M client, an open source C implementation of LwM2M, named Anjay, was used [17]. And for the server side, an open source LwM2M Server implementation, called Leshan, was used [18]. In order to achieve target functionalities, Anjay and Leshan were extended with the proposed Batch object and the reverse interaction model.

In this evaluation, an external system (Localization Server) calculates the position (X,Y,Z) of a device and notifies the device of its new position. The Localization Server holds a LwM2M server, that the positioned device registered, and performs operations on the Position object instance, presented in Table III, available on the positioned devices.

TABLE III
THE LWM2M POSITION OBJECT (ID: 3360).

RID	Name	R/W	Type	Description
5701	Sensor Units	R/W	String	Measurement Units Definition.
5702	X Value	R/W	Float	The measured value along X axis.
5703	Y Value	R/W	Float	The measured value along Y axis.
5704	Z Value	R/W	Float	The measured value along Z axis.
5516	Uncertainty	R/W	Float	The accuracy of the position.
5518	Timestamp	R/W	Time	The time of location measurement.
5750	Application Type	R/W	String	The Application Type.

Since Localization Server calculates coordinates at once, the X Value, Y Value and Z Value (5702, 5703, 5704) resources are updated every time a new position triple is received. For this purpose, two different approaches are evaluated. In the first, each new position is sent over three separate packets (x,y,z). In the second approach, when using the Batch object, after the registration of the LwM2M client registers with the localization server, the localization server creates a Batch object instance and immediately set the Configuration resource value to the corresponding resource IDs in the Position object as illustrated in Figure 5.

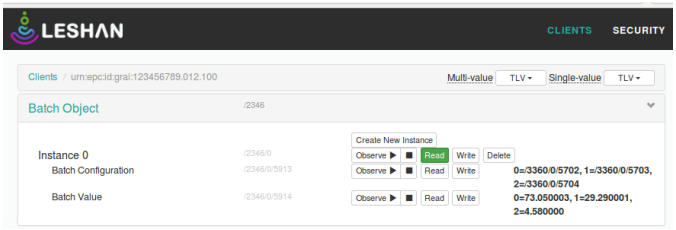

Fig. 5. A sample Batch Object Instance in LwM2M Server (Leshan).

In each approach, the position is updated 10 times. The packets for creating the Position object and creating the Batch object are included in the calculations. The impact of having to create these instances in the first solution becomes smaller over time. The measured data traffic values are provided in Table IV. These values show that Batch approach provides a lot more efficiency than updating the (x, y, z) coordinates separately in three different requests.

TABLE IV
DATA TRAFFIC FOR DIFFERENT POSITION UPDATING POLICIES.

	Batch	**Three req.**
Bytes exchanged	2.065	4.203
Bytes sent	1.403	2.522
Bytes received	662	1.681
Payload sent (in bytes)	899	1.220
Payload received (in bytes)	158	379
Packets exchanged	24	62

3) The combination of Batch and Reverse Interaction Model: Another limitation of LwM2M is that it is not possible to get periodic updates of multiple resources in a single packet, without having to send requests. This is what we attempt to solve by combining the Batch with the reverse interaction model created with the Notify object. In this subsection we evaluate potential performance improvement for such an operation.

In this experiment, an external system wants to know the temperature (/3303/0/5700), humidity (/3304/0/5700), coordinates (X,Y,Z: /3360/0/5702, /3360/0/5703, /3360/0/5704) and battery state (/3/0/9) of a device with a period of 15 minutes. For this purpose, three different approaches are considered. In the first approach, the external system makes six separate requests to the resources every 15 minutes, while the second approach uses the Batch object and the external system sends a single request for the Batch object's value every 15 minutes. The last approach is based on a Notify Object instance which holds a target Resource URI that links to a Batch Object Instance. This operation will create a reverse interaction model which automatically sends updates of the Batch every 15 minutes to the external system.

The results, as presented in Table V, are calculated for 96 updates in total (24h / 15min) and the JSON data format is used. These results show that the Batch is almost six times more efficient than making six separate requests. The reverse interaction model is improving the efficiency even more by sending the data automatically to the registered server.

TABLE V
RESULTS FOR GETTING RESOURCE VALUES OVER MULTIPLE OBJECTS.

	Sep. requests	**Batch**	**Reverse Batch**
Bytes exchanged	95.904	30.259	23.658
Bytes sent	39.168	6.774	173
Bytes received	56.736	23.485	23.458
Payload sent (in bytes)	14.976	2.700	131
Payload received (in bytes)	32.544	19.411	19.411
Packets exchanged	1.152	194	98

D. To BATCH or Not To BATCH

The previous sections show that the Batch object can improve communication efficiency in LwM2M, however there might be cases or scenarios where the Batch approach does not improve the performance, even decrease in certain cases. Therefore, there are certain factors which should be taken into consideration while deciding whether or not to use Batch.

First of all, the number of aggregated resources and their types are crucial factors which define the contribution of the Batch operation. For instance, if total payload size of an aggregated response message exceeds MTU, the performance of Batch operation would decrease.

Secondly, another factor that should be considered is packet loss. Using the Batch requires only a single response, while separate requests require multiple responses. That means, in case of packet loss, all aggregated data will be lost.

In case of observing a Batch object, with every resource in the Batch configuration (even those unchanged) being included in every notification, the packets containing the updated values are larger than needed. This means that, depending on how many resources the CoAP client wants to observe, how often each resource provides an update and how long the continued interest has to be shown, the CoAP client should thus choose between observing multiple resources at a time or observing a Batch object instance.

Moreover, security should also be taken into account. Due to the significant overhead added by security protocols, the probability of exceeding the MTU will be a lot higher and the efficiency of Batch operation might diminish. Of course, security will also impact individual requests, but this trade-off should be studied further.

V. CONCLUSION

Although the LwM2M protocol is providing powerful and efficient functionalities for IoT devices, it has still some limitations. Therefore, in this paper, we propose extensions to the LwM2M protocol in order to improve communication efficiency and introduce Intermittent Connectivity in LwM2M.

The proposed Notify Object enables a Ready-to-Receive functionality, which allows a LwM2M client to notify LwM2M servers that it is ready to receive downlink messages and to keep the network connectivity open if it is necessary for downlink communication (e.g. NAT-connected devices). This object can be also used to create reverse interaction model where LwM2M clients can send periodic updates without any need for a request. In addition, the Batch object allows LwM2M servers to perform actions on multiple resources in a device via a single request. Despite resulting in larger packet sizes, the Batch approach can improve the communication efficiency. However, there are certain factors that should be taken into account in the decision process of the Batch usage: number of resources, resource types, encoding type, update rates, resource data size and network MTU. Such factors affect the efficiency of the Batch operation and the unthoughtful usage of Batch object can lead to performance drops.

ACKNOWLEDGMENT

Part of this research was funded by the Flemish FWO SBO S004017N IDEAL-IoT (Intelligent DEnse And Longe range IoT networks) project, and by the ICON project MAGICIaN. MAGICIaN is realized in collaboration with imec, with project support from VLAIO and Innoviris. Project partners are imec, Orange Belgium, Televic Healthcare, Restore and Citymesh.

REFERENCES

[1] A. Nordrum, "Popular Internet of Things Forecast of 50 Billion Devices by 2020 Is Outdated," August 2016.
[2] "IPv6 over Low power WPAN (6LoWPAN)." http://datatracker.ietf.org/wg/6lowpan, accessed on 1 June, 2018.
[3] "Routing Over Low power and Lossy networks (ROLL)." http://datatracker.ietf.org/wg/roll, accessed on 1 June, 2018.
[4] "Constrained RESTful Environments (CORE)." http://datatracker.ietf.org/wg/core/, accessed on 1 June, 2018.
[5] oneM2M, "White Paper: The interoperability enabler for the entire M2M and IoT ecosystem," January 2015.
[6] Open Mobile Alliance. http://openmobilealliance.org/, accessed on 1 June, 2018.
[7] Internet Protocol for Smart Objects (IPSO) Alliance. https://www.ipso-alliance.org/, accessed on 1 June, 2018.
[8] Open Mobile Alliance, "Lightweight Machine to Machine Technical Specification," February 2017.
[9] Z. Shelby, K. Hartke and C. Bormann, "The Constrained Application Protocol (CoAP)," RFC 7252, IETF, June 2014.
[10] A. Bhattacharyya and S. Bandyopadhyay and A. Pal and T. Bose, "Constrained Application Protocol (CoAP) Option for No Server Response," August 2016.
[11] Open Mobile Alliance, "OMA LightweightM2M (LwM2M) Object and Resource Registry."
[12] J. Jimenez, M. Koster and H. Tschofenig, "IPSO Smart Objects," January 2016.
[13] P. van der Stok, C. Bormann and A. Sehgal, "PATCH and FETCH Methods for the Constrained Application Protocol (CoAP)," April 2017.
[14] N. Sornin, M. Luis, T. Eirich, T. Kramp, O. Hersent, "LoRa specification. Technical report," January 2015.
[15] A. Augustin, J. Yi, T. Clausen, and W. M. Townsley, "A Study of LoRa: Long Range and Low Power Networks for the Internet of Things," *Sensors*, September 2016.
[16] The Things Network, *LoRa(WAN) airtime calculator*. https://docs.google.com/spreadsheets/d/1QvcKsGeTTPpr9icj4XkKXq4r2zTc2j0gsHLrnplzM3I, accessed on 1 June, 2018.
[17] AVSystem, "Anjay: open-source LwM2M Library." https://www.avsystem.com/products/anjay, accessed on 1 June, 2018.
[18] Eclipse, "Leshan: An OMA Lightweight M2M (LWM2M) implementation in Java." https://eclipse.org/leshan/, accessed on 1 June, 2018.